US008074203B2

(12) **United States Patent**
Dye et al.

(10) **Patent No.: US 8,074,203 B2**
(45) **Date of Patent: *Dec. 6, 2011**

(54) **GRAPHICAL PROGRAM EXECUTION WITH DISTRIBUTED BLOCK DIAGRAM DISPLAY**

(75) Inventors: **Robert E. Dye**, Austin, TX (US); **Darshan Shah**, Round Rock, TX (US); **Steve Rogers**, Austin, TX (US); **Greg Richardson**, Round Rock, TX (US); **Dean A. Luick**, Austin, TX (US)

(73) Assignee: **National Instruments Corporation**, Austin, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 742 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **12/143,198**

(22) Filed: **Jun. 20, 2008**

(65) **Prior Publication Data**

US 2009/0024981 A1 Jan. 22, 2009

Related U.S. Application Data

(63) Continuation of application No. 10/772,518, filed on Feb. 5, 2004.

(51) **Int. Cl.**
G06F 9/44 (2006.01)
G06F 15/16 (2006.01)
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **717/113**; 717/110; 717/124; 709/201; 709/203; 715/733; 715/740

(58) **Field of Classification Search** 717/104–113; 709/201–203; 715/733, 740

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,827,404 A 5/1989 Barstow et al.
4,849,880 A 7/1989 Bhaskar et al.
4,901,221 A 2/1990 Kodosky et al.
5,109,504 A 4/1992 Littleton
(Continued)

FOREIGN PATENT DOCUMENTS

EP 0367709 5/1990
(Continued)

OTHER PUBLICATIONS

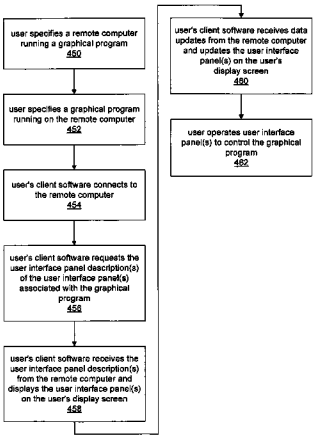
Lee, et al., "Gabriel: A Design Environment for DSP", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, No. 11, Nov. 1989, pp. 1751-1762.
(Continued)

Primary Examiner — Wei Zhen
Assistant Examiner — Qing Chen
(74) *Attorney, Agent, or Firm* — Meyertons Hood Kivlin Kowert & Goetzel, P.C.; Jeffrey C. Hood

(57) **ABSTRACT**

System and method for executing a graphical program on a first computer and providing a user interface of the graphical program on a second computer, where the graphical program includes a plurality of interconnected function icons representing graphical data flow of a function. Information indicating a plurality of graphical programs on the first computer, e.g., a list, may be displayed on the second computer, and user input specifying the graphical program on the first computer received to the second computer, e.g., selecting the graphical program from the list of graphical programs. The graphical program is executed on the first computer, and information describing the user interface of the graphical program is provided to the second computer during execution. The user interface of the graphical program is displayed on the second computer, facilitating interaction between a user of the second computer and the graphical program executing on the first computer.

20 Claims, 12 Drawing Sheets


user specifies a remote computer running a graphical program
460
user specifies a graphical program running on the remote computer
462
user's client software connects to the remote computer
464
user's client software requests the user interface panel description(s) of the user interface panel(s) associated with the graphical program
466
user's client software receives the user interface panel description(s) from the remote computer and displays the user interface panel(s) on the user's display screen
468
user's client software receives data updates from the remote computer and updates the user interface panel(s) on the user's display screen
480
user operates user interface panel(s) to control the graphical program
482

U.S. PATENT DOCUMENTS

5,283,861	A	2/1994	Dangler et al.
5,309,556	A	5/1994	Sismilich
5,377,318	A	12/1994	Wolber
5,437,464	A	8/1995	Terasima et al.
5,481,740	A	1/1996	Kodosky
5,497,498	A	3/1996	Taylor
5,535,342	A	7/1996	Taylor
5,541,849	A	7/1996	Rostoker et al.
5,544,320	A	8/1996	Konrad
5,555,201	A	9/1996	Dangelo et al.
5,566,295	A	10/1996	Cypher et al.
5,583,749	A	12/1996	Tredennick et al.
5,603,043	A	2/1997	Taylor et al.
5,638,299	A	6/1997	Miller
5,652,875	A	7/1997	Taylor
5,652,909	A	7/1997	Kodosky
5,684,980	A	11/1997	Casselman
5,724,074	A	3/1998	Chainani et al.
5,732,277	A	3/1998	Kodosky et al.
5,737,235	A	4/1998	Kean et al.
5,760,788	A	6/1998	Chainani et al.
5,784,275	A	7/1998	Sojoodi et al.
5,801,689	A	9/1998	Huntsman
5,949,412	A	9/1999	Huntsman
6,020,881	A	2/2000	Naughton et al.
6,064,409	A	5/2000	Thomsen et al.
6,102,965	A	8/2000	Dye et al.
6,138,150	A	10/2000	Nichols et al.
6,173,438	B1	1/2001	Kodosky et al.
6,219,628	B1	4/2001	Kodosky et al.
6,226,776	B1	5/2001	Panchul et al.
6,230,307	B1	5/2001	Davis et al.
6,313,851	B1	11/2001	Matthews et al.

FOREIGN PATENT DOCUMENTS

EP	0398646	11/1990
EP	0426909	5/1991
WO	94/10627	5/1994
WO	94/15311	7/1994
WO	96/14618	5/1996

OTHER PUBLICATIONS

Bier, et al., "Gabriel: A Design Environment for DSP", IEEE Micro Issue—vol. 10, No. 5, 1990, pp. 28-45.

Ludolph, et al., "The Fabrik Programming Environment", IEEE, pp. 222-230, 1988.

Stallings, William, "Operating Systems: Multiprocessor and Real-Time Scheduling"; 1995; pp. 394-414; 2nd Ed., Prentice Hall.

Douglas, Bruce D., Ph.D., "Custom Embedded Communication Protocols," Conference Proceedings of Embedded Systems Conference East, Mar. 10-12, 1997, Boston, Massachusetts, pp. 637-657.

Labrosse, Jean, "Inside Real-Time Kernels," Conference Proceedings of Embedded Systems Conference East, Mar. 10-12, 1997, Boston, Massachusetts, pp. 205-214.

Labrosse, Jean, "Designing with Real-Time Kernels," Conference Proceedings of Embedded Systems Conference East, Mar. 10-12, 1997, Boston, Massachusetts, pp. 379-389.

Theimer, et al., "Preemptable Remote Execution Facilities for the V-System," ACM, 1985, pp. 2-12.

Shearman, Sam, "Hardware-in-the-loop connectivity extends continuous-system simulation", Personal Engineering, Jun. 1997, pp. 28-37.

Chen, et al., "Software Environment for WASMII: A Data Driven Machine with a Virtual Hardware," Field Programmable Logic Architectures, Synthesis and Applications, 4th International Workshop on Field-Programmable Logic and Applications, FPL '94 Proceedings, Berlin, Germany, 1994, 12 pages.

Edwards, et al., "Software acceleration using programmable hardware devices," IEEE Proceedings: Computers and Digital Techniques, vol. 143, No. 1, Jan. 1996, pp. 55-63.

Leeser, et al., "High Level Synthesis and Generating FPGAs with the BEDROC System," Journal of VLSI Signal Processing, vol. 6, No. 2, Aug. 1993, pp. 191-214.

Žagar, et al., "CORBA-based Remote Instrumentation"; 9th Mediterranean Electrotechnical Conference (MELECON); May 1998; pp. 1294-1298; vol. 2; Tel-Aviv.

Johnston, et al., "High-Speed Distributed Data Handling for On-Line Instrumentation Systems"; ACM/IEEE 1997 Supercomputing Conference; Aug. 16, 1997; pp. 1-19.

Beguelin, et al., "Visualization and Debugging in a Heterogenous Environment", IEEE, Jun. 1993, pp. 88-95, vol. 26, Issue 6.

Jamal, et al., "The Applicability of the Visual Programming Language LabVIEW to Large Real-World Applications", IEEE, 1995, pp. 99-106.

Wiberg, Per-Arne, "Graphical Programming of Time-Deterministic Real-Time Systems", IEEE, 1996, pp. 173-180.

Ade, et al., "Hardware-Software Codesign with GRAPE," Proceedings of the 6th IEEE International Workshop on Rapid System Prototyping, Jun. 9, 1995, pp. 40-47.

Lauwereins, et al., "Grape-II: A System-Level Prototyping Enrivonment for DSP Applications," IEEE, vol. 28, Issue 2, Feb. 1995, pp. 35-43.

Wenban, et al., "A Software Development System for FPGA-Based Data Acquisition Systems", FPGAs for Custom Computing Machines, IEEE, 1996, 10 pages.

Moulton, et al., "Remote programmability of graphic interactions in a host/satellite configuration," ACM, 1976, pp. 204-211.

Shaheen et al., "Remote Laboratory Experimentation", IEEE, Proceeding of American Control Conference, pp. 1326-1329, Jun. 1998.

Gillet, et al., "Telepresence: An Opportunity to Develop Real-World Experimentation Education"; European Control Conference; Jul. 1997; pp. 1-6; Brussels, Belgium.

Ross, et al., "The design and programming of a display interface system integrating multi-access and satellite computers," 1967, ACM, pp. 1-14.

De Coster, "Grape-II: An Introduction", Automatic Control and Computer Architectures Department, Katholieke Universiteit Leuven, Belgium, Feb. 22, 1996 [retrieved Oct. 6, 1999], retrieved from the Internet: http://www.esat.kuleuven.ac.be/acca, 25 pages.

Collamati, et al., "Induction Machine Stator Fault On-Line Diagnosis Based on LabVIEW Environment", Mediterranean Electrotechnical Conference, vol. 1, p. 495-498, May 1996.

Spoelder, et al., "Virtual Instrumentation: A Survey of Standards and Their Interrelation", Proc. IEEE Instr. and Measurement Tech. Conf., vol. 1, pp. 676-681, May 1997.

Srinivasan, et al., "LabVIEW Program Design for On-Line Data Acquisition and Predictive Maintenance", Proc. Of the 30th Southeastern Symposium on System Theory, pp. 520-524, Mar. 1998.

Wahidanabanu, et al., "Virtual Instrumentation with Graphical Programming for Enhanced Detection and Monitoring of Partial Discharges", Proc. Electrical Insulation Conf. 1997, pp. 291-296, Sep. 1997.

"Choosing Block-Diagram Tools for DSP Design", Berkeley Design Technology, http://www.bdti.com/articles/info_dspmt95blockdiagram.htm, May 9, 2003, pp. 1-7.

"Real-Time Workshop for Use with SIMULINK: User's Guide," The MathWorks, Inc., May 1994, 229 pages.

"Guide to Rapid Prototyping with Simulink, Real-Time Workshop and dSPACE," The MathWorks, 1995, 16 pages.

"Real-Time Interface to Simulink, RTI 30, User's Guide," dSPACE, 1995, 125 pages.

Gorman, et al., "Real-Time Data Acquisition and Controls Using MatLAB", Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium (ASME), 1995, 4 pages.

"SPW—MatLAB Co-Simulation Interface Product Data Sheet," Alta Group, 1996, 2 pages.

"Signal Processing WorkSystem, MATLAB Interface User's Guide," Alta Group, Oct. 1995, 72 pages.

"Alta Group of Cadence Design Systems, Inc.," Berkeley Design Technology, Inc., 1995, 34 pages.

"Code Generation System (CGS) Product Data Sheet," Alta Group, 1994, 8 pages.

"SPW/CGS Porting Kits Product Data Sheet," Alta Group, 1994, 2 pages.

"MultiProx for SPW Product Data Sheet," Alta Group, 1994, 4 pages.
"DSP ProCoder for SPW Product Data Sheet," Alta Group, 1994, 4 pages.
"Xanalog Corporation Sales Manual," Xanalog Corporation, Jan. 1987, 8 pages.
"Available XA-1000 Literature and Its Use," Xanalog Corporation, 1986, 2 pages.
"Xanalog XA-1000 Programming ICONS," Xanalog Corporation, 1986, 15 pages.
"Xanalog's CAE System: The Fastest AT Alive," Mass High Tech, vol. 4, No. 22, Aug. 1988, 1 page.
"Xanalog: The Computer Aided Engineering Workstation Comes to Simulation," Simulation vol. 47, No. 1, Jul. 1986, 3 pages.
"Xanalog/RT Real Time Analog and Digital I/O," Xanalog Corporation, 1990, 4 pages.
"Xanalog/SC+", Xanalog Corporation, Sep. 1990, 4 pages.
"Xanalog: Specializing in Workstations for Continuous Dynamic Simulation," Xanalog Corporation, 1987, 24 pages.
"Xanalog Real-Time User Guide," Xanalog Corporation, 1994, 28 pages.
Lee, et al., "Gabriel: A Design Environment for Programmable DSPs", Nov. 7, 1988, 13 pages.
Lee, et al., "A Design Tool for Hardware and Software for Multiprocessor DSP Systems", May 1989, 4 pages.
"Gabriel 0.7 Overview," The University of California, 1990, 5 pages.
Buck, et al., "Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model", Institute of Electrical and Electronics Engineers, http://www.synopsys.com/, 1993, 4 pages.
Lee, "Design Methodology for DSP", Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1992, 4 pages.
Pino, et al., "Interface Synthesis in Heterogeneous System-Level DSP Design Tools", Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, May 1996, 4 pages.
Pino, Jose Luis, "Software Synthesis for Single-Processor DSP Systems Using Ptolemy: Master's Report," Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, May 1993, 48 pages.
Kalavade, et al., "A Hardware-Software Codesign Methodology for DSP Applications", Institute of Electrical and Electronics Engineers, Sep. 1993, 12 pages.
Pino, et al., "Automatic Code Generation for Heterogeneous Multiprocessors", Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1994, 4 pages.
Bindra, Ashok, "Tool chest continues to grow", Electronic Engineering Times, Dec. 15, 1995, 2 pages.
Pino, et al., "Mapping Multiple Independent Synchronous Dataflow Graphs onto Heterogeneous Multiprocessors", Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Oct. 1994, 6 pages.
Kalavade, et al., "Hardware/Software Co-Design Using Ptolemy—A Case Study", Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Sep. 1992, 18 pages.
Pino, et al., "Software Synthesis for DSP Using Ptolemy", Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Journal of VLSI Signal Processing, 9, 7-21, 1995, 15 pages.
Bhattacharyya, et al., "The Almagest: Ptolemy 0.7 User's Manual," vol. 1, 1997, 532 pages.
"i-Logix Product Overview," iLogix, 1996, 44 pages.
Press Release, "i-Logix Statemate MAGNUM Supports PCs", iLogix, Jan. 31, 1997, 2 pages.
Press Release, "i-Logix Signs Reseller Agreement for Virtual Prototypes, Inc.'s VAPS Product Line", Feb. 11, 1997, 2 pages.
Press Release, "i-Logix Introduces Rhapsody, Object-Oriented Analysis, Design and Implementation Tool", Feb. 10, 1997, 2 pages.
"Statement/C Product Overview," iLogix, 1995, 4 pages.
Press Release, "i-Logix and Integrated Systems Link Statemate MAGNUM and MATRIX AutoCode", Jan. 3, 1997, 2 pages.
Press Release, "i-Logix and Wind River unveil Industry's First Rapid Prototyping Solution for Testing Embedded Systems at ESC West in San Jose", Sep. 17, 1996, 3 pages.
Press Release, "i-Logix Inc. Endorses Unified Modeling Language", Jan. 16, 1997, 1 page.
Lapsley, Philip Dean , "Host Interface and Debugging of Dataflow DSP Systems", Master's Thesis, 1991, pp. 1-55.
"LabVIEW User Manual for Windows"; Sep. 1994; pp. 4/1-4/21; National Instruments Corporation; Austin, TX.

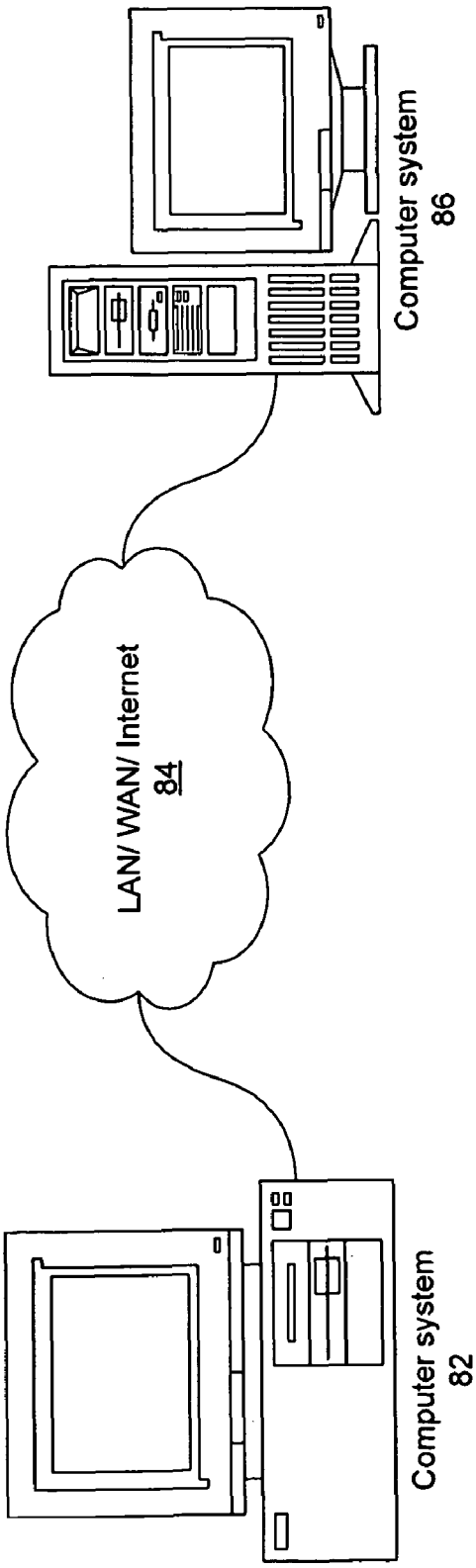


FIG. 1

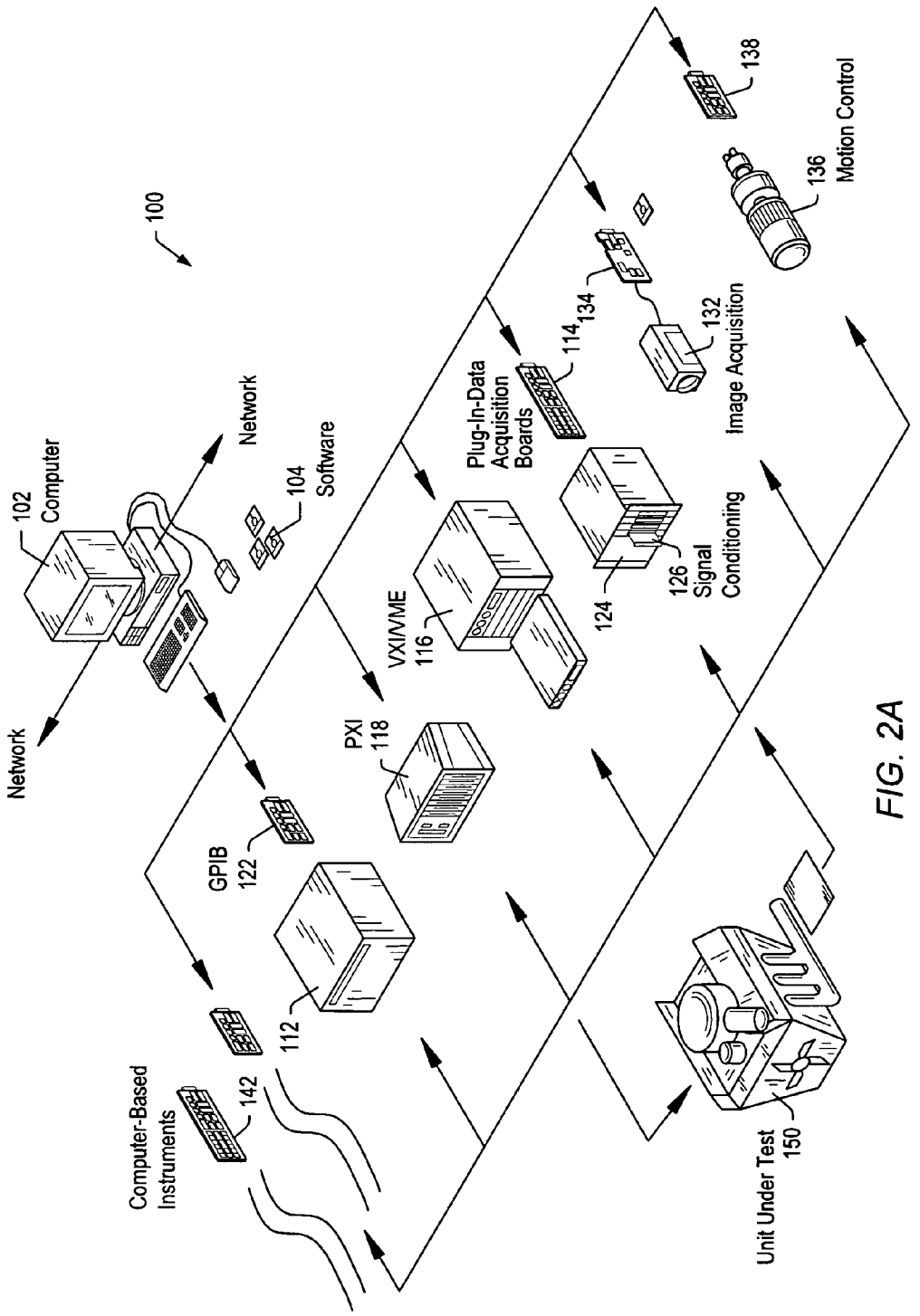

100
102 Computer
Network
104 Software
Network
Plug-In-Data Acquisition Boards
114
134
132 Image Acquisition
136 Motion Control
138
VXI/VME
116
124
126 Signal Conditioning
PXI
118
GPIB
122
112
Computer-Based Instruments
142
Unit Under Test
150

FIG. 2A

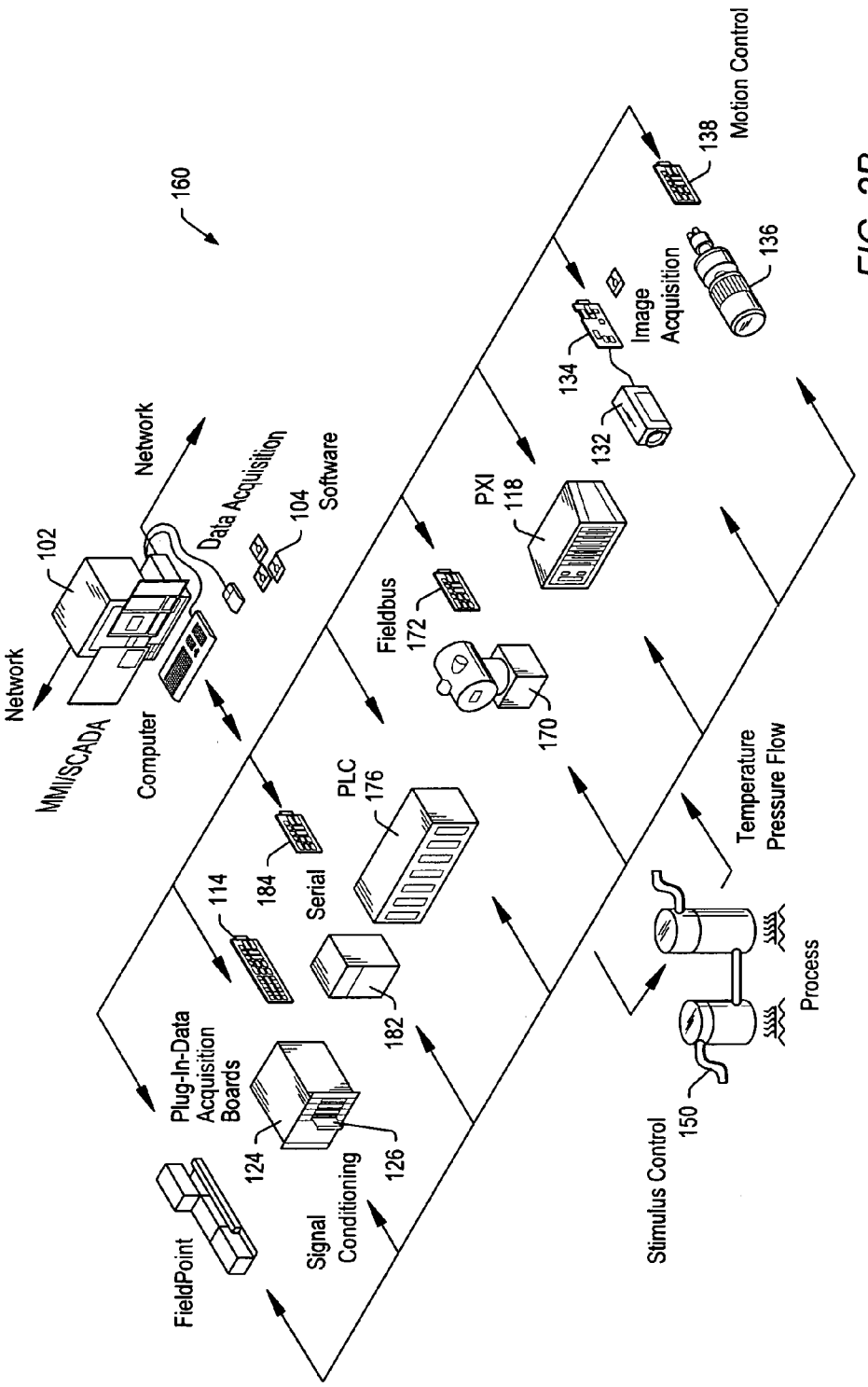

160
Network
102
Data Acquisition
104
Software
Network
MMI/SCADA
Computer
PXI
118
132
134
Image Acquisition
136
138
Motion Control
Fieldbus
172
170
PLC
176
114
184
Serial
182
Plug-In-Data Acquisition Boards
124
126
Signal Conditioning
FieldPoint
Stimulus Control
150
Temperature
Pressure Flow
Process

FIG. 2B

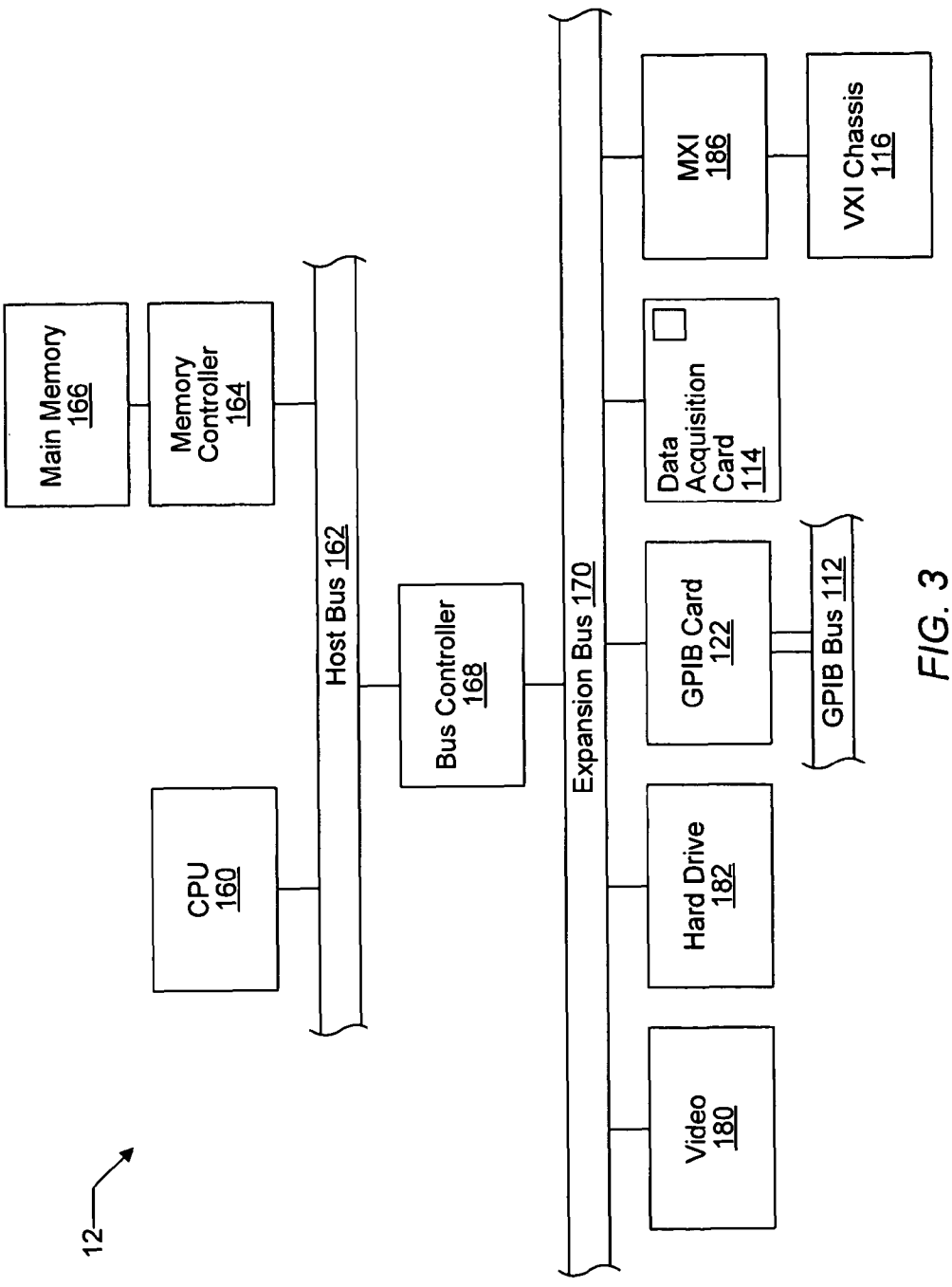

12
CPU 160
Main Memory 166
Memory Controller 164
Host Bus 162
Bus Controller 168
Expansion Bus 170
Video 180
Hard Drive 182
GPIB Card 122
GPIB Bus 112
Data Acquisition Card 114
MXI 186
VXI Chassis 116

FIG. 3

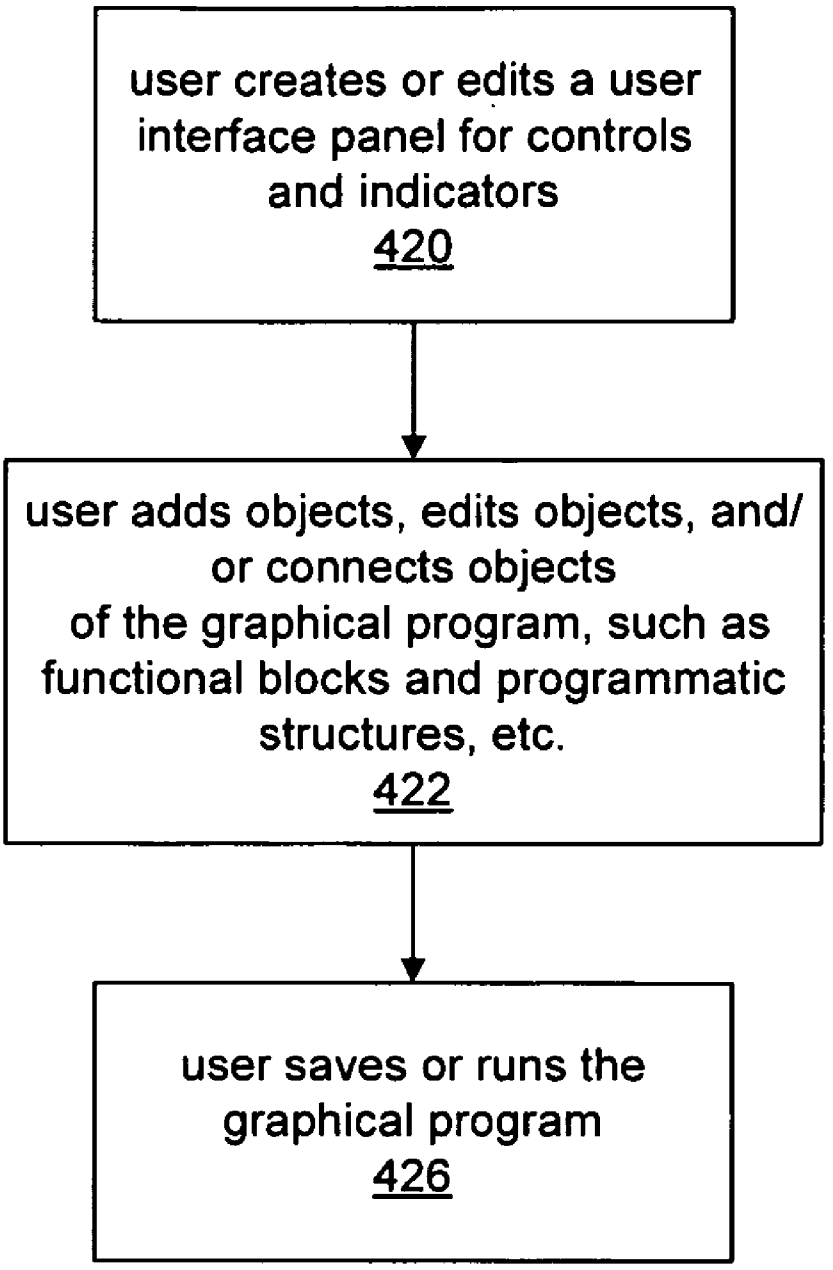

user creates or edits a user interface panel for controls and indicators
420
user adds objects, edits objects, and/or connects objects of the graphical program, such as functional blocks and programmatic structures, etc.
422
user saves or runs the graphical program
426

FIG. 4

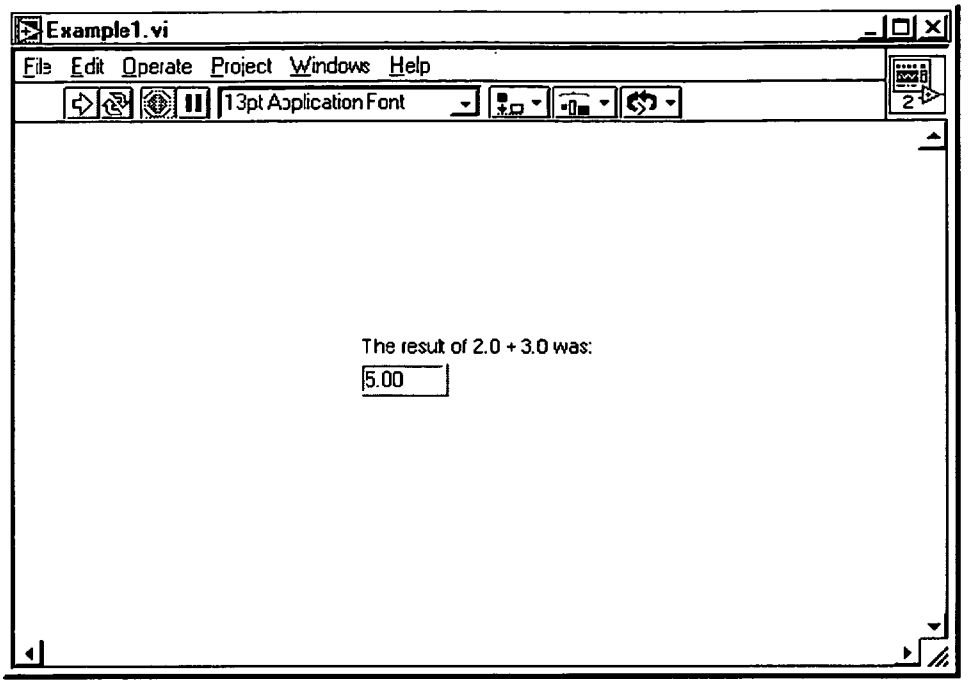

Example1.vi
File Edit Operate Project Windows Help
13pt Application Font
The result of 2.0 + 3.0 was:
5.00

FIG. 5

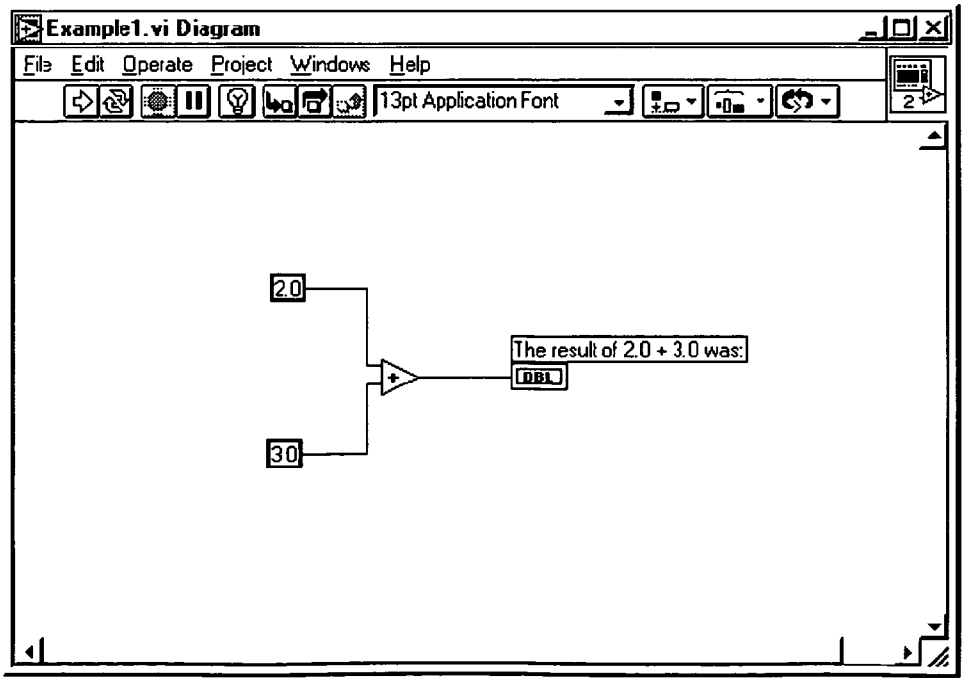

Example1.vi Diagram
File Edit Operate Project Windows Help
13pt Application Font
2.0
30
The result of 2.0 + 3.0 was:
DBL

FIG. 6

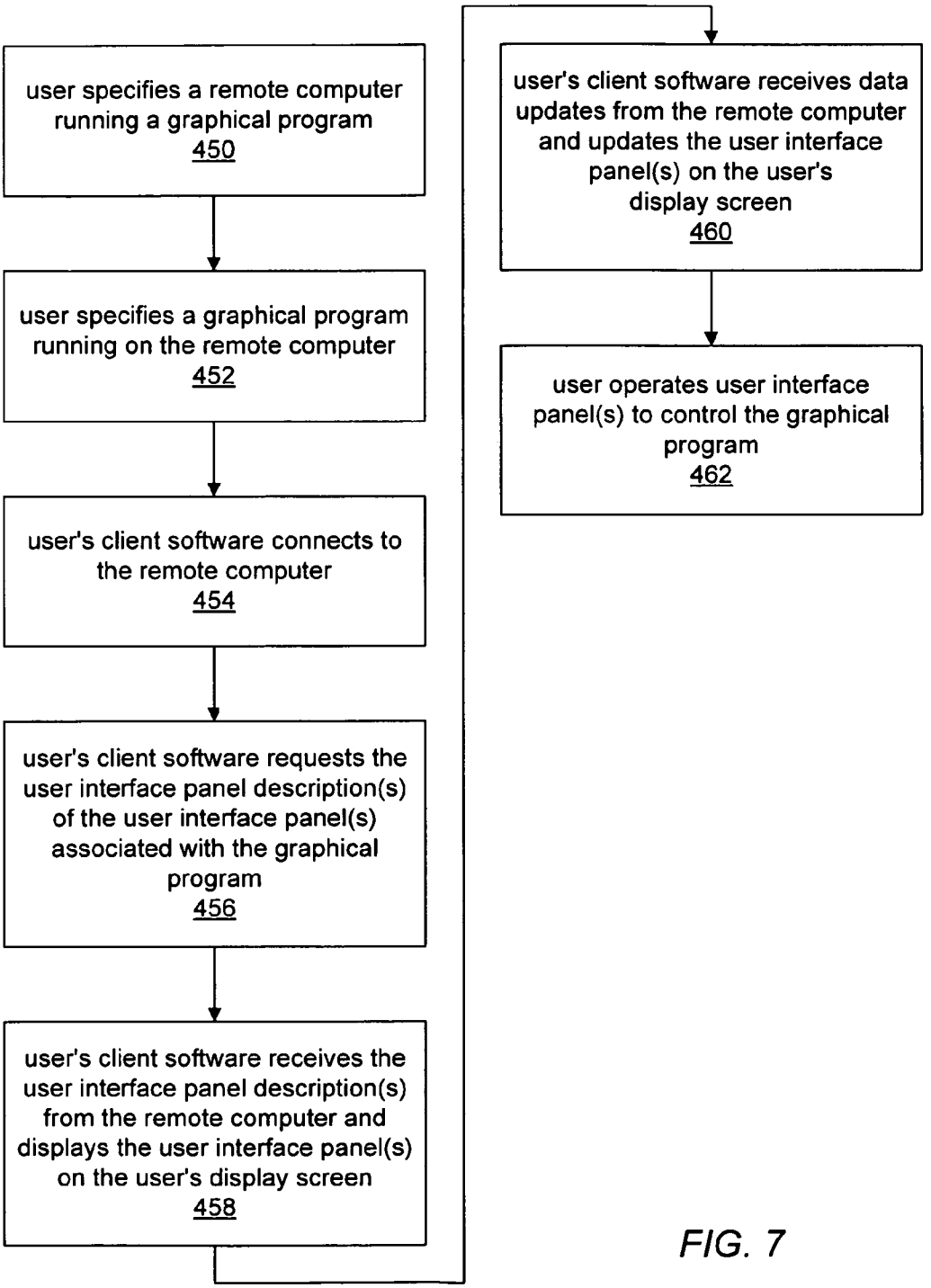

user specifies a remote computer running a graphical program
450
user specifies a graphical program running on the remote computer
452
user's client software connects to the remote computer
454
user's client software requests the user interface panel description(s) of the user interface panel(s) associated with the graphical program
456
user's client software receives the user interface panel description(s) from the remote computer and displays the user interface panel(s) on the user's display screen
458
user's client software receives data updates from the remote computer and updates the user interface panel(s) on the user's display screen
460
user operates user interface panel(s) to control the graphical program
462

FIG. 7

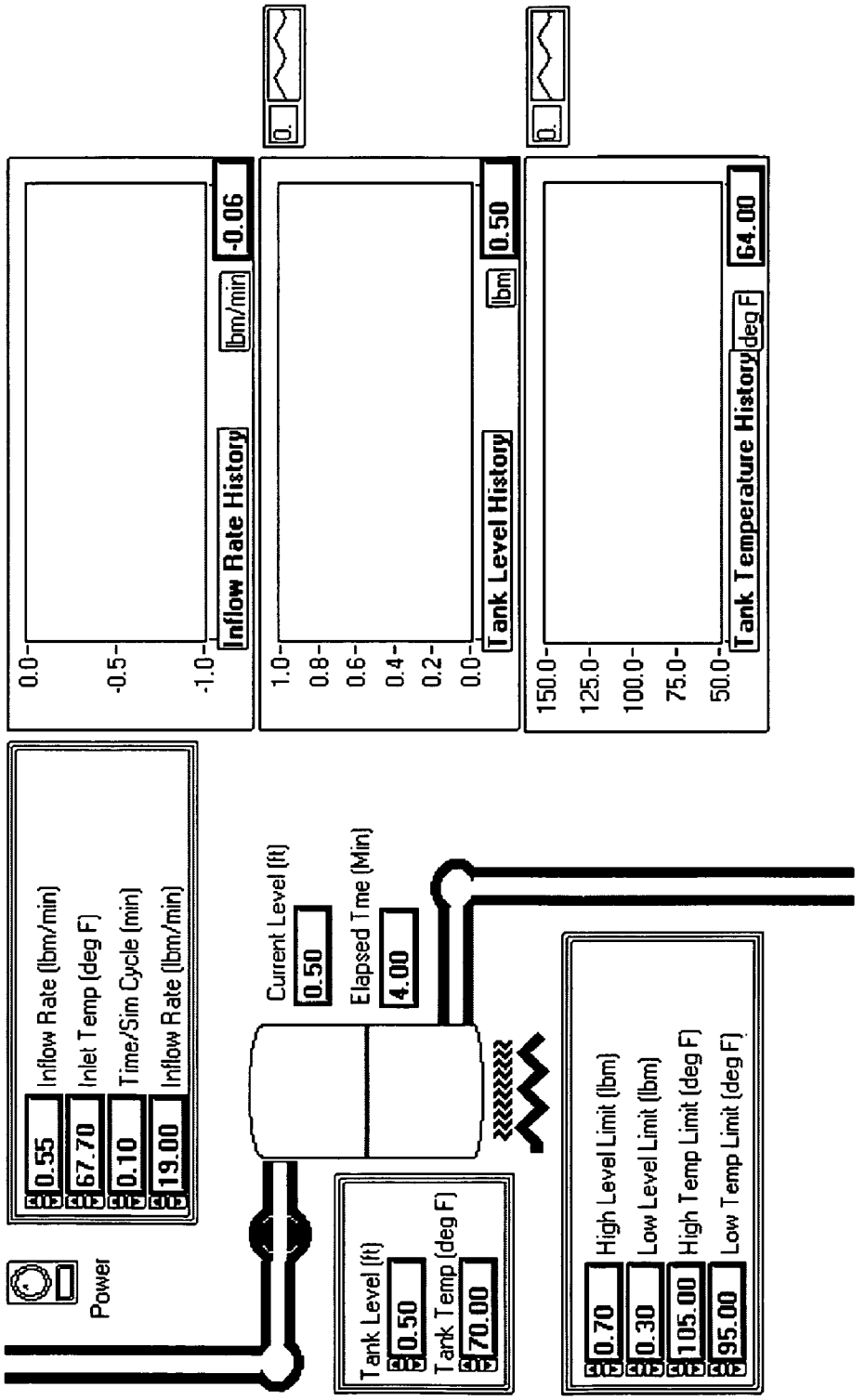

Inflow Rate History
lbm/min
-0.06
0.0
-0.5
-1.0
Tank Level History
lbm
0.50
1.0
0.8
0.6
0.4
0.2
0.0
Tank Temperature History
deg F
64.00
150.0
125.0
100.0
75.0
50.0
Power
Inflow Rate (lbm/min)
0.55
Inlet Temp (deg F)
67.70
Time/Sim Cycle (min)
0.10
Inflow Rate (lbm/min)
19.00
Current Level (ft)
0.50
Elapsed Time (Min)
4.00
Tank Level (ft)
0.50
Tank Temp (deg F)
70.00
High Level Limit (lbm)
0.70
Low Level Limit (lbm)
0.30
High Temp Limit (deg F)
105.00
Low Temp Limit (deg F)
95.00

FIG. 8A

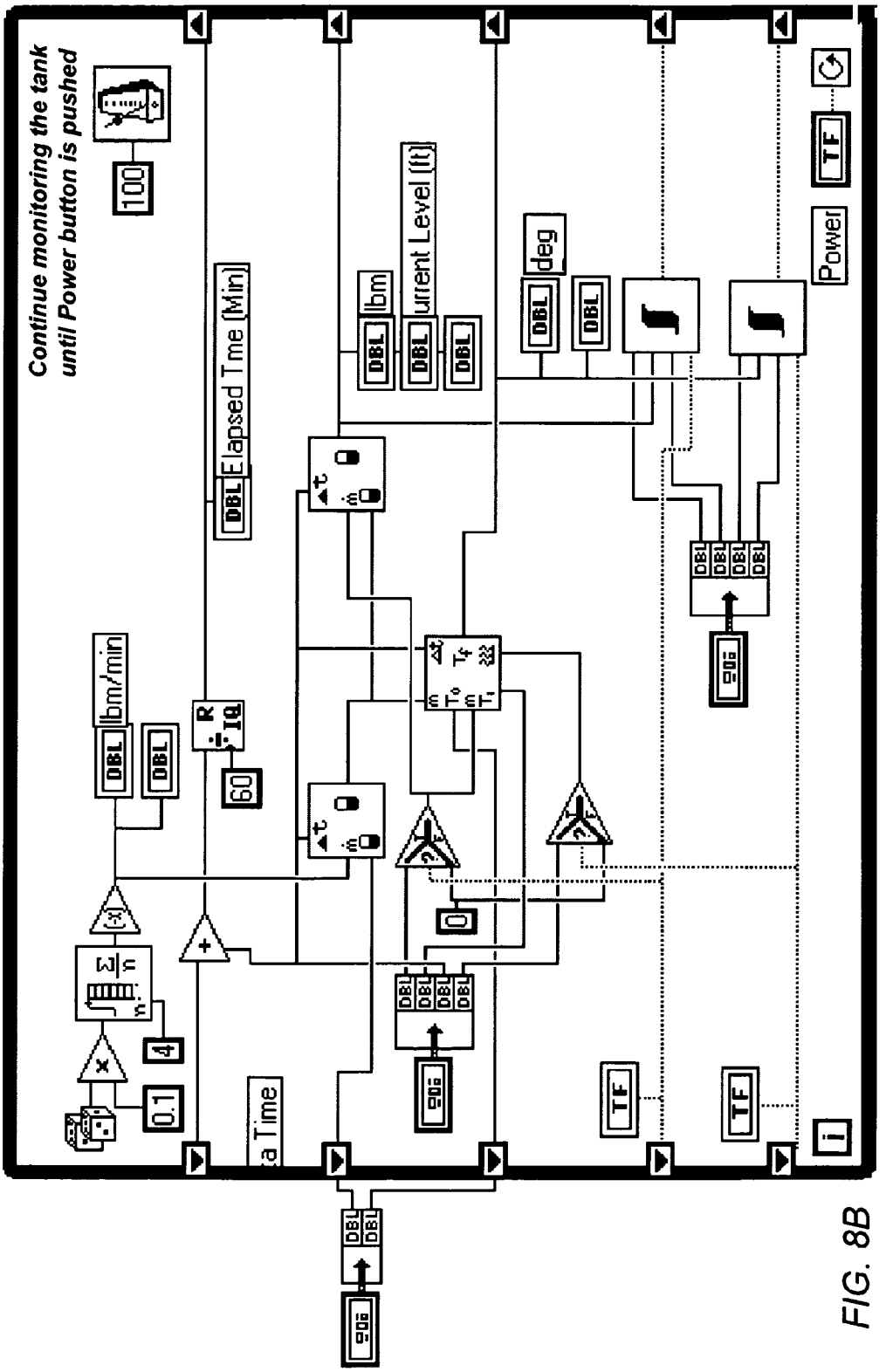

Continue monitoring the tank until Power button is pushed
100
Elapsed Tme (Min)
lbm
Current Level (ft)
deg
Power
TF
lbm/min
60
0.1
4
0
a Time
i

FIG. 8B

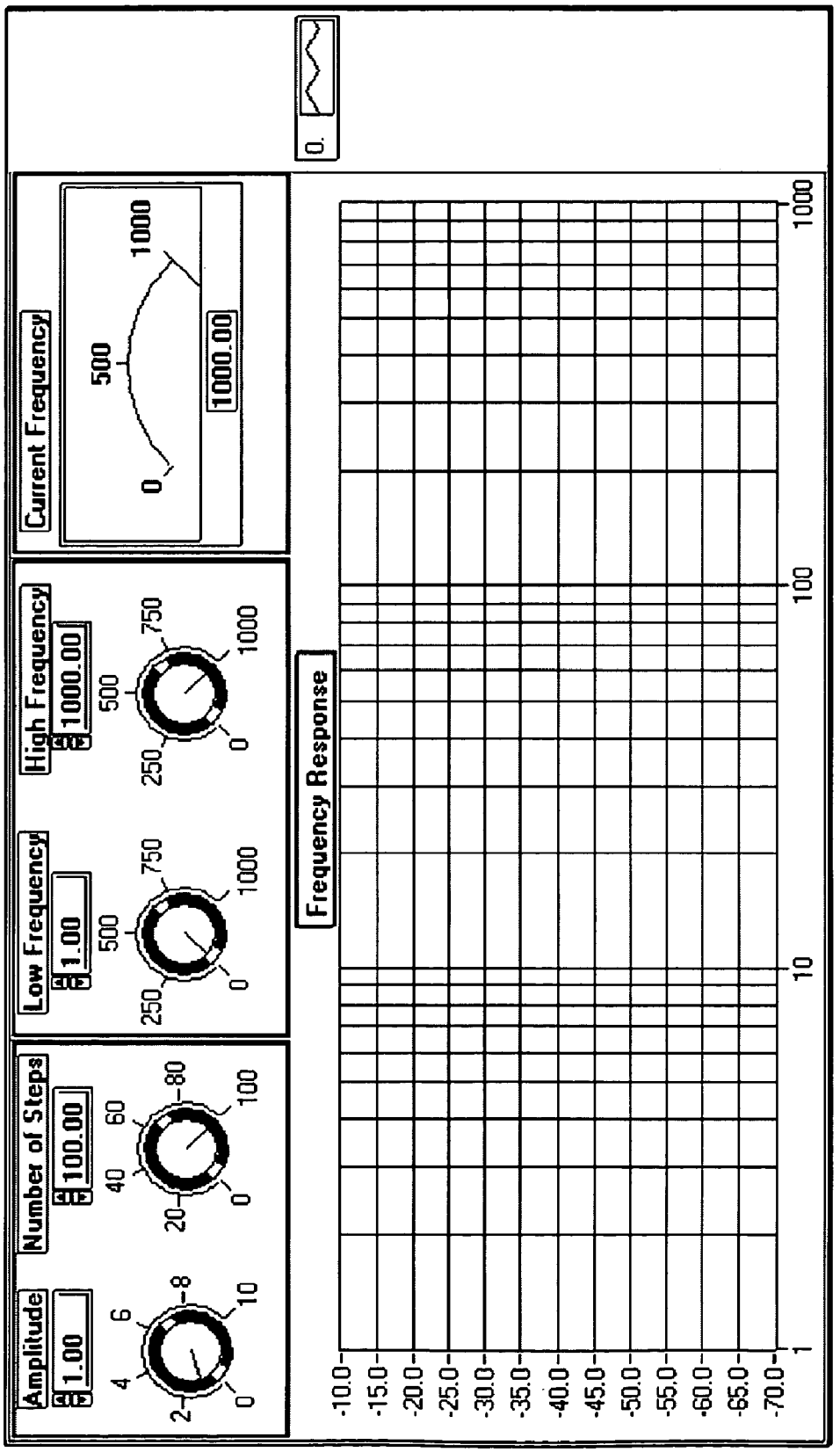

Amplitude
1.00
Number of Steps
100.00
Low Frequency
1.00
High Frequency
1000.00
Current Frequency
1000.00
Frequency Response
-10.0
-15.0
-20.0
-25.0
-30.0
-35.0
-40.0
-45.0
-50.0
-55.0
-60.0
-65.0
-70.0
1
10
100
1000

FIG. 9A

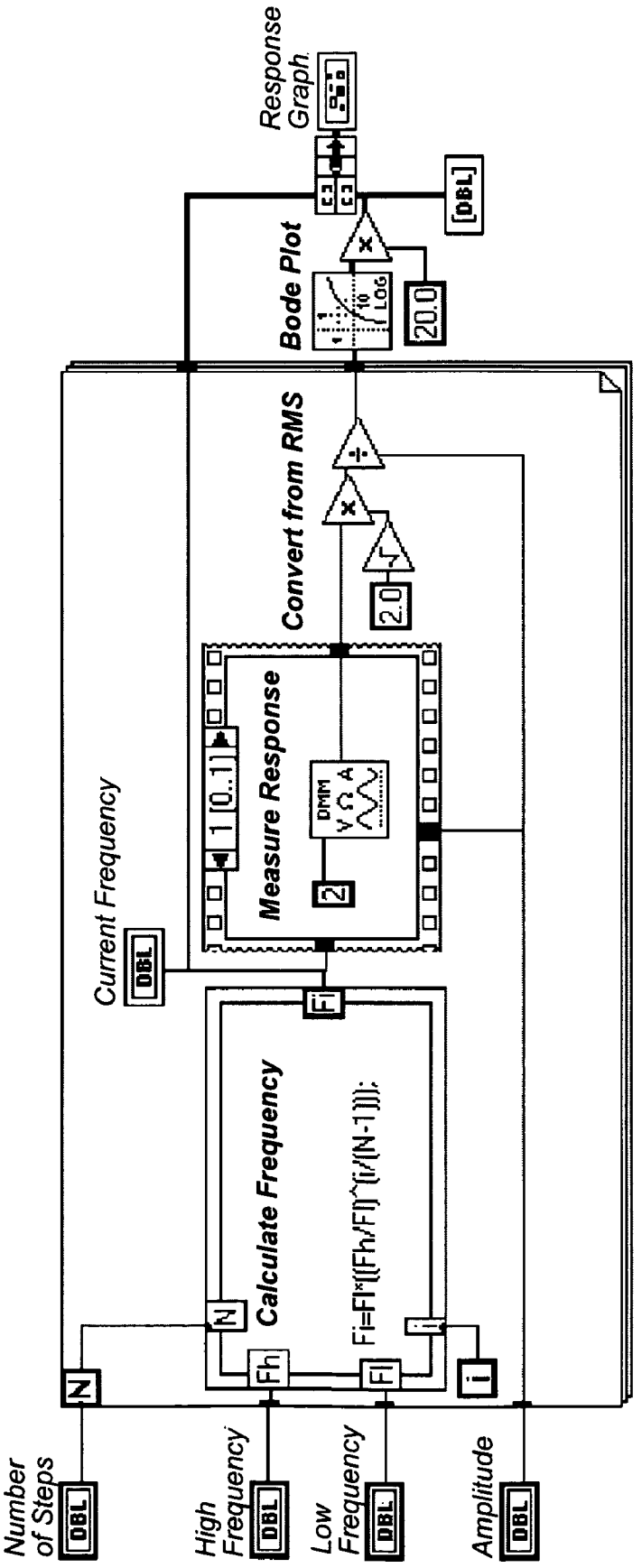

Response Graph
Bode Plot
20.0
DBL
Convert from RMS
2.0
Measure Response
1 [0..1]
DMM
2
Current Frequency
DBL
Calculate Frequency
N
Fh
Fl
Fi=Fl*([Fh/Fl]^[i/(N-1)]);
Fi
i
N
Number of Steps
DBL
High Frequency
DBL
Low Frequency
DBL
Amplitude
DBL

FIG. 9B

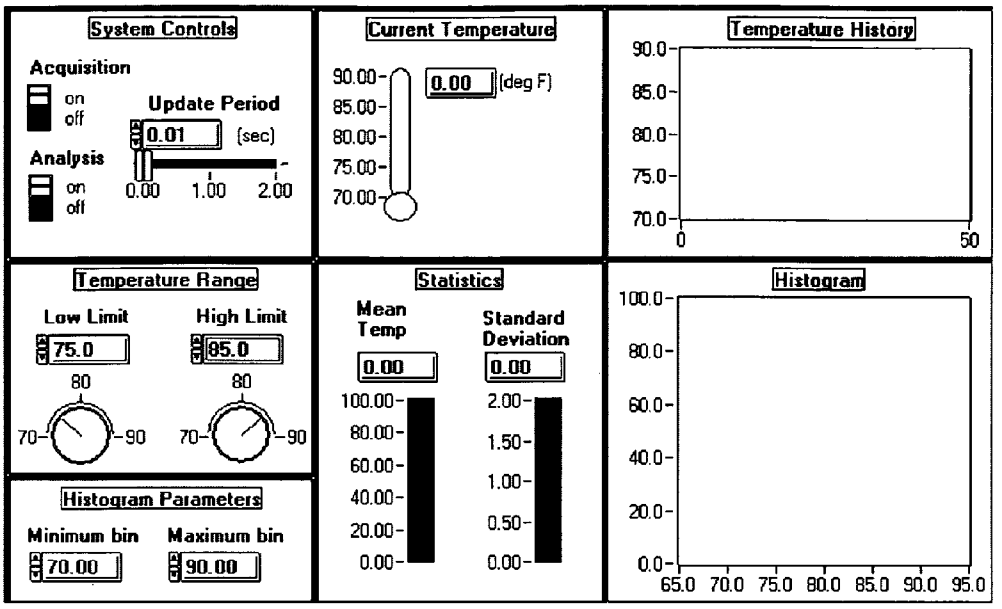


FIG. 10A

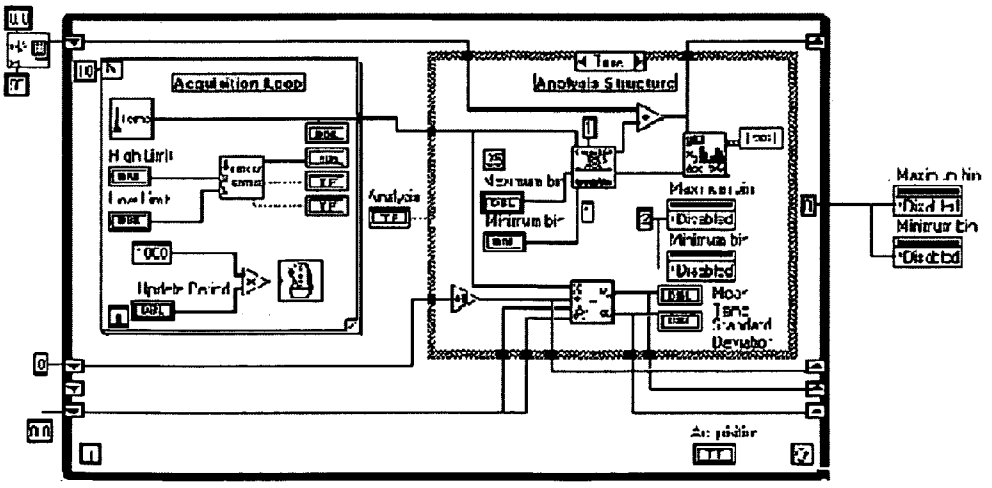


FIG. 10B

GRAPHICAL PROGRAM EXECUTION WITH DISTRIBUTED BLOCK DIAGRAM DISPLAY

CONTINUATION AND PRIORITY DATA

This application is a continuation of U.S. patent application Ser. No. 10/772,518 titled "Graphical Programming System with Block Diagram Execution and Distributed User Interface Display" filed Feb. 5, 2004, whose inventors were Robert Dye, Darshan Shah, Steve Rogers, Greg Richardson, and Dean A. Luick, which claims benefit of priority of U.S. patent application Ser. No. 09/617,600 titled "Graphical Programming System with Distributed Block Diagram Execution and User Interface Display" filed Jun. 13, 2000, whose inventors were Robert Dye, Darshan Shah, Steve Rogers, Greg Richardson, and Dean A. Luick, which claims benefit of priority of U.S. provisional application Ser. No. 60/149,950 titled "Graphical Programming System with Distributed Block Diagram Execution and Front Panel Display" filed Aug. 19, 1999, whose inventors were Robert Dye, Darshan Shah, Steve Rogers, and Greg Richardson.

FIELD OF THE INVENTION

The present invention relates to the field of graphical programming and virtual instrumentation. In particular, the invention relates to a system and method wherein a graphical program block diagram executes on a server computer, and one or more client computers receive and display a graphical program user interface panel corresponding to the block diagram, wherein the graphical program user interface panel can be used to provide input to or display output to from the block diagram. The present invention further relates to a distributed virtual instrumentation system, wherein a block diagram executes on a server computer and one or more front panels are displayed on client computers.

DESCRIPTION OF THE RELATED ART

Traditionally, high level text-based programming languages have been used by programmers in writing application programs. Many different high level text-based programming languages exist, including BASIC, C, FORTRAN, Pascal, COBOL, ADA, APL, etc. Programs written in these high level languages are translated to the machine language level by translators known as compilers or interpreters. The high level text-based programming languages in this level, as well as the assembly language level, are referred to as text-based programming environments.

Increasingly, computers are required to be used and programmed by those who are not highly trained in computer programming techniques. When traditional text-based programming environments are used, the user's programming skills and ability to interact with the computer system often become a limiting factor in the achievement of optimal utilization of the computer system.

There are numerous subtle complexities which a user must master before he can efficiently program a computer system in a text-based environment. The task of programming a computer system to model a process often is further complicated by the fact that a sequence of mathematical formulas, mathematical steps or other procedures customarily used to conceptually model a process often does not closely correspond to the traditional text-based programming techniques used to program a computer system to model such a process. In other words, the requirement that a user program in a text-based programming environment places a level of abstraction between the user's conceptualization of the solution and the implementation of a method that accomplishes this solution in a computer program. Thus, a user often must substantially master different skills in order to both conceptually model a system and then to program a computer to model that system. Since a user often is not fully proficient in techniques for programming a computer system in a text-based environment to implement his model, the efficiency with which the computer system can be utilized to perform such modeling often is reduced.

Examples of fields in which computer systems are employed to model and/or control physical systems are the fields of instrumentation, process control, industrial automation, and simulation. Computer modeling or control of devices such as instruments or industrial automation hardware has become increasingly desirable in view of the increasing complexity and variety of instruments and devices available for use. However, due to the wide variety of possible testing/control situations and environments, and also the wide array of instruments or devices available, it is often necessary for a user to develop a program to control a desired system. As discussed above, computer programs used to control such systems had to be written in conventional text-based programming languages such as, for example, assembly language, C, FORTRAN, BASIC, or Pascal. Traditional users of these systems, however, often were not highly trained in programming techniques and, in addition, traditional text-based programming languages were not sufficiently intuitive to allow users to use these languages without training. Therefore, implementation of such systems frequently required the involvement of a programmer to write software for control and analysis of instrumentation or industrial automation data. Thus, development and maintenance of the software elements in these systems often proved to be difficult.

U.S. Pat. Nos. 4,901,221; 4,914,568; 5,291,587; 5,301,301; and 5,301,336; among others, to Kodosky et al disclose a graphical system and method for modeling a process, i.e., a graphical programming environment, which enables a user to easily and intuitively model a process. The graphical programming environment disclosed in Kodosky et al can be considered the highest and most intuitive way in which to interact with a computer. A graphically based programming environment can be represented at a level above text-based high level programming languages such as C, Pascal, etc. The method disclosed in Kodosky et al allows a user to construct a diagram using a block diagram editor, such that the diagram created graphically displays a procedure or method for accomplishing a certain result, such as manipulating one or more input variables to produce one or more output variables. In response to the user constructing a data flow diagram or graphical program using the block diagram editor, data structures may be automatically constructed which characterize an execution procedure which corresponds to the displayed procedure. The graphical program may be compiled or interpreted by a computer using these data structures. Therefore, a user can create a computer program solely by using a graphically based programming environment. This graphically based programming environment may be used for creating virtual instrumentation systems, industrial automation systems, modeling processes, and simulation, as well as for any type of general programming.

Therefore, Kodosky et al teaches a graphical programming environment wherein a user places or manipulates icons in a block diagram using a block diagram editor to create a graphical "program." A graphical program for controlling or modeling devices, such as instruments, processes or industrial automation hardware, is referred to as a virtual instrument

(VI). In creating a virtual instrument, a user may create a front panel or user interface panel. The front panel includes various front panel objects, such as controls or indicators, that represent or display the respective input and output that will be used by the graphical program or VI, and may include other icons which represent devices being controlled. When the controls and indicators are created in the front panel, corresponding icons or terminals may be automatically created in the block diagram by the block diagram editor. Alternatively, the user can place terminal icons or input/output blocks in the block diagram which may cause the display of corresponding front panel objects in the front panel, either at edit time or at run time.

During creation of the graphical program, the user selects various functions that accomplish his desired result and connects the function icons together. For example, the functions may be connected in a data flow and/or control flow format. The functions may be connected between the terminals of the respective controls and indicators. For example, the user may create or assemble a data flow program, referred to as a block diagram, representing the graphical data flow which accomplishes his desired function. The assembled graphical program may then be compiled or interpreted to produce machine language that accomplishes the desired method or process as shown in the block diagram.

A user may input data to a virtual instrument using front panel controls. This input data propagates through the data flow block diagram or graphical program and appears as changes on the output indicators. In an instrumentation application, the front panel can be analogized to the front panel of an instrument. In an industrial automation application the front panel can be analogized to the MMI (Man Machine Interface) of a device. The user may adjust the controls on the front panel to affect the input and view the output on the respective indicators. Alternatively, the front panel may be used merely to view the input and output, and the input may not be interactively manipulable by the user during program execution.

Thus, graphical programming has become a powerful tool available to programmers. Graphical programming environments such as the National Instruments LabVIEW product have become very popular. Tools such as LabVIEW have greatly increased the productivity of programmers, and increasing numbers of programmers are using graphical programming environments to develop their software applications. In particular, graphical programming tools are being used for test and measurement, data acquisition, process control, man machine interface (MMI), supervisory control and data acquisition (SCADA) applications, simulation, and machine vision applications, among others.

In many scenarios, it would be desirable to further separate the user interface panel, also referred to above as the front panel, of a graphical program from the block diagram of the graphical program. For example, a user developing an instrumentation application, such as a test and measurement application or a process control application, may desire the graphical program to execute on a computer located in a laboratory or manufacturing facility, but may want to interact with the program by viewing the program's user interface panel from another computer, such as a workstation located in the user's office. As another example, a program developer may construct a graphical program and desire to enable others to interact with or view the results of the program. For example, the program developer may desire to enable multiple Internet users to connect to the computer running the graphical program and view the graphical program's user interface.

It would thus be desirable to provide a general system and method for enabling various types of graphical programs having various types of user interface panels to export their user interface panels as described above, with a minimal amount of programming effort. It may also be desirable to provide the above capabilities using common networking and software standards so that users working on various types of computing platforms could connect to the remote computer running the graphical program, view the user interface panel of the graphical program, and possibly also use the user interface panel to remotely use or control the graphical program. It may also be desirable to require users to install a minimal amount of client software in order to gain these abilities, and/or to enable the necessary client software to be automatically downloaded and installed.

SUMMARY OF THE INVENTION

The problems outlined above may in large part be solved by providing a system and method enabling distributed display of the user interface of a graphical program executing on a server computer. In one embodiment, the system includes a server computer where a graphical program executes, and one or more client computers connected to the server computer which receive and display a user interface, e.g., one or more user interface panels, corresponding to the graphical program. In one embodiment, the user interface can be used from the client computer(s) to provide input to or display output from the graphical program during program execution. In one specific embodiment, the invention may comprise a distributed virtual instrumentation system, wherein a graphical program executes on a server computer to perform a measurement or automation function, and one or more front panels are displayed on client computers, thus enabling one or more users to remotely view and/or control the measurement or automation function.

In one embodiment, a user of a client computer specifies a remote server computer on which a graphical program executes. The remote server information may be specified in various ways. For example, the information may be specified as a uniform resource locator (URL), as an internet protocol (IP) address, as a machine name and TCP/IP port number, etc. In one embodiment, a user may specify the remote computer by entering a URL into an application such as a web browser or other application with web-browsing functionality. As described below, the application may include a protocol handler plug-in enabled to process the URL and connect to the remote computer.

When the user specifies the remote computer running the graphical program, the user may also specify the particular graphical program desired. For example, a parameter indicating the name of the graphical program may be appended to the URL, etc. The user may also specify the remote computer without also specifying the particular graphical program. For example, the remote computer may comprise a web server. The user may enter the URL of a web page associated with the web server, and the web server may return a list of graphical programs running on the remote computer. The user may then select one or more graphical programs from this list. The user's client software is operable to then display the user interface panels associated with the selected graphical program(s) on the user's display screen.

In one embodiment, the user's client software comprises a web browser (or application with web-browsing functionality) with a plug-in operable to communicate with the remote graphical program. In this embodiment, the plug-in may display the user interface panel directly in the web browser's

window. The user's client software preferably communicates with an agent or software program running on the remote computer using a communication protocol based on the standard TCP/IP protocol. When the user specifies the remote computer for a connection, the agent on the remote computer transfers a description of the graphical program's user interface panel to the user's client software. This description may be sent in the same format used to store the user interface panel information on the remote computer. The user interface panel description may, of course, be sent in various other formats, e.g., as an XML description. The user's client-side software, e.g., web browser plug-in, is preferably enabled to interpret any type of user interface panel description that it may receive from the remote computer, and is enabled to appropriately display the user interface panel to the user.

Once the graphical program's user interface panel is received and displayed on the user's display screen, the user interface panel may be dynamically updated during execution of the graphical program block diagram. For example, the user interface panel may include a graph which displays various types of measurement data produced by the block diagram, such as an electrical signal, meteorological data, etc., and this graph may scroll on the user's display as the measured data values change in response to graphical program execution. As another example, the user interface panel may comprise numerical text indicators that are updated with new values periodically, etc.

The user may also interact with the user interface panel on the client computer to provide input to the block diagram executing on the server computer, e.g. by issuing standard point-and-click type GUI commands. The user's input is passed to the remote graphical program on the server computer, and the graphical program responds accordingly. In other words, the user may interact with the remote graphical program exactly as he would interact with the program if it were running locally on the user's computer. A means for coordinating control among users may be included so that multiple users interacting with the same graphical program do not interfere with each others' actions.

As described below, in one embodiment, a user may also request and receive the remote graphical program's block diagram, e.g., to edit or debug the graphical program.

As noted above, in the preferred embodiment, a TCP/IP-based communication protocol is used for communication between the user's client software and the remote server computer executing the graphical program. In an alternative embodiment, the DataSocket system and method, disclosed in U.S. patent application Ser. No. 09/185,161, may be used to facilitate the communication between the user's client software and the remote computer running the graphical program. The DataSocket system comprises a client software component that addresses data sources/targets using a URL, much the way that a URL is used to address web pages anywhere in the world.

In one embodiment, the remote graphical program executes within a graphical programming environment including functionality referred to as "VI Server". VI Server functionality may be used to enable user clients to connect to and interact with a remote graphical program. For more information on VI Server, please refer to the patent applications incorporated by reference below.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

FIG. **1** illustrates a computer system connected through a network to a second computer system;

FIGS. **2**A and **2**B illustrate representative instrumentation and process control systems including various I/O interface options;

FIG. **3** is a block diagram of the computer system of FIGS. **1**, **2**A and **2**B;

FIG. **4** is a flowchart diagram illustrating one embodiment of interactively creating or editing a graphical program;

FIGS. **5** and **6** illustrate a simple graphical program comprising a user interface panel and a block diagram; and

FIG. **7** is a flowchart diagram illustrating one embodiment of a user accessing a remote graphical program.

FIGS. **8-10** illustrate exemplary graphical programs and their associated user interfaces.

While the invention is susceptible to various modifications and alternative forms specific embodiments are shown by way of example in the drawings and are herein described in detail. It should be understood however, that drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed. But on the contrary the invention is to cover all modifications, equivalents and alternative following within the spirit and scope of the present invention as defined by the appended claims.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Incorporation by Reference

The following references are hereby incorporated by reference in their entirety as though fully and completely set forth herein.

U.S. patent application Ser. No. 08/912,445 titled "Embedded Graphical Programming System" filed Aug. 18, 1997.

U.S. patent application Ser. No. 08/810,079 titled "System and Method for Developing Automation Clients Using a Graphical Data Flow Program" filed Mar. 4, 1997.

U.S. patent application Ser. No. 08/916,005 titled "System and Method for Providing Client/Server Access to Graphical Programs" filed Aug. 21, 1997.

U.S. patent application Ser. No. 09/136,123 titled "System and Method for Accessing Object Capabilities in a Graphical Program" filed Aug. 18, 1998.

U.S. patent application Ser. No. 09/518,492 titled "System and Method for Programmatically Creating a Graphical Program" filed Mar. 3, 2000.

U.S. patent application Ser. No. 09/185,161 titled "Data Socket System and Method for Accessing Data Sources Using URLs" filed Nov. 3, 1998, whose inventor was Paul F. Austin.

U.S. patent application Ser. No. 09/374,740 titled "System and Method for Automatically Creating URLs for Accessing Data Sources and Data Targets" filed Aug. 13, 1999, whose inventors were Paul Austin, David Fuller, Kurt M. Carlson, Chris Mayer, Stephen Rogers, Joe Savage, and Brian Sierer.

U.S. patent application Ser. No. 09/546,047 titled "System and Method for Connecting to and Viewing Live Data using a Standard User Agent" filed Apr. 10, 2000, whose inventor was Paul Austin.

U.S. provisional application Ser. No. 60/149,950 titled "Graphical Programming System with Distributed Block Diagram Execution and Front Panel Display" filed Aug. 19, 1999, whose inventors were Robert Dye, Darshan Shah, Steve Rogers, and Greg Richardson.

U.S. patent application Ser. No. 09/617,600 titled "Graphical Programming System with Distributed Block Diagram Execution and User Interface Display" filed Jun. 13, 2000, whose inventors were Robert Dye, Darshan Shah, Steve Rogers, Greg Richardson, and Dean A. Luick.

U.S. patent application Ser. No. 10/772,518 titled "Graphical Programming System with Block Diagram Execution and Distributed User Interface Display" filed Feb. 5, 2004, whose inventors were Robert Dye, Darshan Shah, Steve Rogers, Greg Richardson, and Dean A. Luick.

FIG. **1**—Computer System Connected to a Network

FIG. **1** illustrates an exemplary computer network in which a computer system **82** is connected through a network **84** to a second computer system **86**. The computer system **82** and the second computer system **86** can be any of various types, as desired. The network **84** can also be any of various types, including a LAN (local area network), WAN (wide area network), or the Internet, among others.

A user of computer system **82** may connect to computer system **86**, according to the system and method described herein. Computer system **82**, which may be referred to as client computer system **82**, comprises client software enabled to receive a description of a graphical program user interface panel and display the panel on the display screen of computer system **82**. For example the client software may comprise a web browser with a web browser plug-in. For example, the web browser may be the Microsoft Internet Explorer web browser, and the plug-in may be constructed according to Microsoft's Asynchronous Pluggable Protocols specification.

Computer system **86**, which may be referred to as server computer system **86**, comprises a graphical program, as well as server-side programs or agents enabling the user of computer system **82** to communicate with computer system **86** according to the present invention. For example, computer system **86** may include VI Server functionality, as discussed above.

Although, only one client is shown connected to computer system **86**, as described above, multiple clients may connect to computer **86** in order to view the graphical program's user interface panel and/or interact with the graphical program. Computer system **86** preferably includes a mechanism for coordinating control of the graphical program among multiple remote users. For example, computer system **86** may distribute control of the graphical program among the users using various methods or algorithms, such as a round-robin scheme, prioritized round-robin scheme, etc. Various types of privileges or permissions may be assigned to different users, granting them different levels of control over the graphical program. For example, the program creator may be authorized to assume complete control over the program, locking out other users. Other users may only be authorized to view the graphical program's user interface panel, but not to use it to control the graphical program, e.g., these users may not be allowed to provide input to the graphical program.

FIGS. **2**A and **2**B—Instrumentation and Industrial Automation Systems

FIGS. **2**A and **2**B illustrate exemplary systems that may store or use programs according to the present invention. These exemplary systems illustrate systems specialized for instrumentation, process control, or other purposes. FIGS. **2**A and **2**B illustrate exemplary server computer systems. Thus, the server computer **86** described above may be comprised in an instrumentation or industrial automation system, wherein the present invention allows for distributed control of a test or automation application. The present invention may of course be used in other types of applications as desired.

FIG. **2**A illustrates an instrumentation control system **100**. The system **100** comprises a host computer **86** (server computer **86**) which connects to one or more instruments. The host computer **86** comprises a CPU, a display screen, memory, and one or more input devices such as a mouse or keyboard as shown. The computer **86** connects through the one or more instruments to analyze, measure, or control a unit under test (WUT) or process **150**.

The one or more instruments may include a GPIB instrument **112** and associated GPIB interface card **122**, a data acquisition board **114** and associated signal conditioning circuitry **124**, a VXI instrument **116**, a PXI instrument **118**, a video device **132** and associated image acquisition card **134**, a motion control device **136** and associated motion control interface card **138**, and/or one or more computer based instrument cards **142**, among other types of devices.

The GPIB instrument **112** is coupled to the computer **86** via the GPIB interface card **122** provided by the computer **86**. In a similar manner, the video device **132** is coupled to the computer **86** via the image acquisition card **134**, and the motion control device **136** is coupled to the computer **86** through the motion control interface card **138**. The data acquisition board **114** is coupled to the computer **86**, and may interface through signal conditioning circuitry **124** to the UUT. The signal conditioning circuitry **124** preferably comprises an SCXI (Signal Conditioning eXtensions for Instrumentation) chassis comprising one or more SCXI modules **126**.

The GPIB card **122**, the image acquisition card **134**, the motion control interface card **138**, and the DAQ card **114** are typically plugged in to an I/O slot in the computer **86**, such as a PCI bus slot, a PC Card slot, or an ISA, EISA or Micro-Channel bus slot provided by the computer **86**. However, these cards **122**, **134**, **138** and **114** are shown external to computer **86** for illustrative purposes.

The VXI chassis or instrument **116** is coupled to the computer **86** via a VXI bus, MXI bus, or other serial or parallel bus provided by the computer **86**. The computer **86** preferably includes VXI interface logic, such as a VXI, MXI or GPIB interface card (not shown), which interfaces to the VXI chassis **116**. The PXI chassis or instrument is preferably coupled to the computer **86** through the computer's PCI bus.

A serial instrument (not shown) may also be coupled to the computer **86** through a serial port, such as an RS-232 port, USB (Universal Serial bus) or IEEE 1394 or 1394.2 bus, provided by the computer **86**. In typical instrumentation control systems an instrument will not be present of each interface type, and in fact many systems may only have one or more instruments of a single interface type, such as only GPIB instruments.

The instruments are coupled to the unit under test (WUT) or process **150**, or are coupled to receive field signals, typically generated by transducers. The system **100** may be used in a data acquisition and control application, in a test and measurement application, a process control application, or a man-machine interface application.

FIG. **2**B illustrates an exemplary industrial automation system **160**. The industrial automation system **160** is similar to the instrumentation or test and measurement system **100** shown in FIG. **2**A. Elements which are similar or identical to elements in FIG. **2**A have the same reference numerals for convenience. The system **160** comprises a computer **86** which connects to one or more devices or instruments. The computer **86** comprises a CPU, a display screen, memory, and one or more input devices such as a mouse or keyboard as shown. The computer **86** connects through the one or more devices to a process or device **150** to perform an automation function,

such as MMI (Man Machine Interface), SCADA (Supervisory Control and Data Acquisition), portable or distributed data acquisition, process control, advanced analysis, or other control.

The one or more devices may include a data acquisition board **114** and associated signal conditioning circuitry **124**, a PXI instrument **118**, a video device **132** and associated image acquisition card **134**, a motion control device **136** and associated motion control interface card **138**, a fieldbus device **170** and associated fieldbus interface card **172**, a PLC (Programmable Logic Controller) **176**, a serial instrument **182** and associated serial interface card **184**, or a distributed data acquisition system, such as the Fieldpoint system available from National Instruments, among other types of devices.

The DAQ card **114**, the PXI chassis **118**, the video device **132**, and the image acquisition card **136** are preferably connected to the computer **86** as described above. The serial instrument **182** is coupled to the computer **86** through a serial interface card **184**, or through a serial port, such as an RS-232 port, provided by the computer **86**. The PLC **176** couples to the computer **86** through a serial port, Ethernet port, or a proprietary interface. The fieldbus interface card **172** is preferably comprised in the computer **86** and interfaces through a fieldbus network to one or more fieldbus devices. Each of the DAQ card **114**, the serial card **184**, the fieldbus card **172**, the image acquisition card **134**, and the motion control card **138** are typically plugged in to an I/O slot in the computer **86** as described above. However, these cards **114**, **184**, **172**, **134**, and **138** are shown external to computer **86** for illustrative purposes. In typical industrial automation systems a device will not be present of each interface type, and in fact many systems may only have one or more devices of a single interface type, such as only PLCs. The devices are coupled to the device or process **150**.

Referring again to FIGS. **2**A and **2**B, the server computer system **86** preferably includes a memory medium on which one or more computer programs or software components according to the present invention are stored. The term "memory medium" is intended to include an installation medium, e.g., a CD-ROM, floppy disks **104**, or tape device, a computer system memory or random access memory such as DRAM, SRAM, EDO RAM, Rambus RAM, etc., or a non-volatile memory such as a magnetic media, e.g., a hard drive, or optical storage. The memory medium may comprise other types of memory as well, or combinations thereof.

In addition, the memory medium may be located in a first computer in which the programs are executed, or may be located in a second different computer which connects to the first computer over a network, such as the Internet. In the latter instance, the second computer provides the program instructions to the first computer for execution. The server computer system **86** may take any of various forms. In a similar manner, the client computer system **82** may take any of various forms, including a personal computer system, workstation, network appliance, Internet appliance, personal digital assistant (PDA), television system or other device. In general, the term "computer system" can be broadly defined to encompass any device having at least one processor which executes instructions from a memory medium.

In one embodiment, the memory medium of the server computer **86** stores software programs for communicating with the client computer system **82**, according to the present invention. For example, the server computer **86** may store network communication software, e.g., TCP/IP software, and may also store application-level software, such as a graphical programming system enabled to communicate with remote computers.

In one embodiment, the memory medium of the client computer **82** stores software programs for communicating with the server computer system **86**, according to the present invention. For example, the client computer **82** may store a standard user agent, such as a web browser or other application with web-browsing functionality, and possibly a specialized browser plug-in for communicating with the server computer.

In one embodiment, the graphical program that users may remotely view or control is a program for data acquisition/generation, analysis, and/or display, or for controlling or modeling instrumentation or industrial automation hardware. For example, in the preferred embodiment, the graphical program is a program constructed using the National Instruments LabVIEW graphical programming environment application, which provides specialized support for developers of instrumentation and industrial automation applications.

However, it is noted that the present invention can be used for a plethora of applications and is not limited to instrumentation or industrial automation applications. In other words, FIGS. **2**A and **2**B are exemplary only, and users may remotely interact with graphical programs for any of various types of purposes in any of various applications.

FIG. **3**—Computer System Block Diagram

FIG. **3** is a block diagram of the computer system illustrated in FIGS. **1**, **2**A and **2**B. It is noted that any type of computer system configuration or architecture can be used as desired, and FIG. **3** illustrates a representative PC embodiment. It is also noted that the computer system may be a general purpose computer system as shown in FIGS. **2**A and **2**B, a computer implemented on a VXI card installed in a VXI chassis, a computer implemented on a PXI card installed in a PXI chassis, or other types of embodiments. The elements of a computer not necessary to understand the present invention have been omitted for simplicity.

The computer **86** (or **82**) includes at least one central processing unit or CPU **160** which is coupled to a processor or host bus **162**. The CPU **160** may be any of various types, including an x86 processor, e.g., a Pentium class, a PowerPC processor, a CPU from the SPARC family of RISC processors, as well as others. Main memory **166** is coupled to the host bus **162** by means of memory controller **164**.

The main memory **166** stores computer programs according to the present invention. The main memory **166** also stores operating system software as well as the software for operation of the computer system, as well known to those skilled in the art. The computer programs of the present invention will be discussed in more detail below.

The host bus **162** is coupled to an expansion or input/output bus **170** by means of a bus controller **168** or bus bridge logic. The expansion bus **170** is preferably the PCI (Peripheral Component Interconnect) expansion bus, although other bus types can be used. The expansion bus **170** includes slots for various devices such as the data acquisition board **114** (of FIG. **2**A), a GPIB interface card **122** which provides a GPIB bus interface to the GPIB instrument **112** (of FIG. **2**A), and a VXI or MXI bus card **186** coupled to the VXI chassis **116** for receiving VXI instruments. The computer **86** further comprises a video display subsystem **180** and hard drive **182** coupled to the expansion bus **170**.

FIGS. **4**-**6**: Interactive Creation of a Graphical Program by a User

FIG. **4** is a flowchart diagram illustrating one embodiment of how a user may interactively or manually create or edit a graphical program. As shown in the flowchart and described below, the user interactively adds various objects to a graphi-

cal program, connects them together, etc. It is noted that the various steps of FIG. **4** may be performed in various orders, or omitted as desired.

In the embodiment shown in FIG. **4**, the steps are performed by a developer creating or editing a graphical program in a graphical programming environment. As shown, in step **420** the developer may create or edit a user interface panel for displaying a graphical user interface. The user interface panel may comprise controls for accepting user input, displaying information such as program output, or both. For example, the user interface panel may include buttons, selectable lists, text boxes, graph controls, images, etc. A developer may "drop" various controls or other objects onto the user interface panel, e.g., by selecting the desired control from a control palette. FIG. **5** illustrates a simple user interface panel. Step **420** is not necessarily performed. For example, a user interface panel may not be desired, a user interface panel may be inherently specified during creation of the block diagram, or a user interface panel may automatically be created as the user creates the executable portions of the graphical program.

In step **422** the developer creates or edits the executable portion of the graphical program, which may referred to as a block diagram. A graphical program may include a block diagram comprising objects referred to herein as "nodes" which are connected together to model the program execution logic, data flow and/or control flow. A block diagram node may be displayed as an icon representing the type or functionality of the node. FIG. **6** illustrates a simple block diagram. As a developer adds objects to the user interface panel, the graphical programming environment may automatically create a corresponding object on the block diagram. Such block diagram nodes which correspond to user interface panel objects are referred to herein as user interface nodes or terminals. For example, the FIG. **6** block diagram node labeled "The result of 2.0+3.0 was:" is a user interface node corresponding to the FIG. **5** user interface output indicator. User interface nodes may be connected with other objects or nodes in the block diagram to participate in the program logic and data/control flow. User interface nodes may map input/output between a user interface panel and a block diagram. For example, the user interface node in FIG. **6** receives data and displays the data in the corresponding user interface indicator in FIG. **5**.

In step **422** of FIG. **4**, the developer adds other objects/nodes to or edits other objects/nodes of the graphical program. These objects or nodes may include function nodes which perform predefined functional operations such as numeric functions, Boolean functions, string functions, array functions, error functions, file functions, application control functions, etc. For example the block diagram shown in FIG. **6** uses an addition function node to add two constants together. In step **422** the developer may also add other types of nodes to the graphical program. For example, nodes may be added which represent numeric constants. FIG. **6** illustrates numeric constant nodes representing the floating point constants 2.0 and 3.0.

Other types of nodes which may be added include subprogram nodes for calling a graphical subprogram, global or local variable nodes for defining and using variables, etc. In step **422**, the developer may also add other types of objects to the graphical program. For example, objects representing programmatic structures such as for loops, while loops, case structures, etc. may be added. The developer may add nodes and other types of objects to a graphical program in various ways, e.g., by selecting a node or object from a palette that displays icons representing the various nodes and objects.

In step **422** of FIG. **4**, the developer may also connect or "wire" the graphical program objects in order to achieve the desired executable logic, data flow, and/or control flow. For example the objects may include input and output terminals, and the developer may connect the output terminal of one node to the input terminal of another node, etc. FIG. **6** illustrates one embodiment of how objects may be connected. In this example, output terminals of the two numeric constant nodes are connected to the input terminals of an addition function node. The addition function node performs the addition operation on the numeric input. The output terminal of the addition function node is connected to the input of the user interface indicator node so that the result of the addition operation is displayed in the user interface panel shown in FIG. **5**.

Programmatic structure objects may also include terminals which integrate them with the other objects of the graphical program. For example, a while loop may comprise a condition terminal to which an output terminal of a node supplying a Boolean value may be connected to signify when the loop should end.

For more information on one embodiment of creating or editing a graphical program, please see the various LabVIEW User and Developer manuals, and LabVIEW version 5.1, available from National Instruments Corporation, which are hereby incorporated by reference.

In step **426** of FIG. **4**, the developer saves or runs the graphical program. The graphical program may be saved in any of various formats. For example, a tree of data structures may be built which represents the various elements of the graphical program and the relationships among the elements, and the data structures may be saved in a binary or text format. These data structures may be compiled into machine code, or interpreted during execution. If the graphical program includes user interface panels, these panels may also be saved. In step **426** the developer may also execute the graphical program. The developer may run the graphical program in any of various ways. For example, a graphical programming environment may allow a program to be run from within the development environment, or the developer may create a standalone program and run the program, etc.

It is noted that steps **420** through **426** typically occur in an iterative manner and typically occur in various orders. For example a developer may add a user interface control to a user interface panel, then connect a user interface node corresponding to the control to another node, then add and connect a function node to the program, then run the program to test it, then change the way a node is connected, etc. Also, as noted above, step **420** may be automatically (e.g., programmatically) performed in response to step **422**. In addition, the user interface panel may be automatically created at edit time, or may be automatically generated at run time. Thus, the flowchart of FIG. **4** is exemplary, and various steps may be combined, omitted, added, or modified as required or desired for developing different graphical programs or using different embodiments of graphical program development environments.

FIG. **7**—Accessing a Remote Graphical Program

FIG. **7** is a flowchart diagram illustrating one embodiment of a user accessing a remote graphical program. In alternative embodiments, various steps of FIG. **7** may be combined, altered, omitted, or may occur in different orders.

As shown, in step **450** of FIG. **7**, a user specifies a remote computer. In step **452**, the user specifies a graphical program on the remote computer. Steps **450** and **452** may be combined into a single step. As discussed above, steps **450** and **452** may be accomplished in any of various ways. For example, the

remote computer and/or the remote graphical program may be implicitly specified by a user specifying a URL which references the remote computer or the remote graphical program. Note that steps **450** and **452** are not necessarily performed directly by a user, but may also be performed programmatically. For example, a user may operate an application that provides a reference to a remote computer and remote graphical program to client software running on the user's machine, which is described below.

In the preferred embodiment, the user performs steps **450** and **452** by interacting with standard, commonly-available client software, such as a web browser or an application including web-browsing functionality, e.g., an application using the Microsoft Internet Explorer code base. For example, the user may provide a URL to the browser application, and the browser application may then contact a web server and receive a list of graphical programs running on the web server computer or another computer. The user may then select one or more of these graphical programs, e.g. by clicking on a hypertext link, etc. Selecting a graphical program may then cause the user's browser application to invoke a browser plug-in to handle the remaining steps of FIG. **7**.

Other embodiments of steps **450-452** are also contemplated. For example, the user may still work within the context of a web browser environment, but may not interact with a web server at any point. For example, the user may provide a URL to the web browser, wherein the URL comprises a protocol scheme which is not natively supported by the web browser. In response, the web browser may delegate the URL to a protocol handler plug-in. For example, such a protocol handler plug-in may be constructed according to the Microsoft Asynchronous Pluggable Protocols specification. The plug-in may then directly contact the remote computer comprising the resource, e.g. graphical program, that the URL references and may continue with steps of FIG. **7**.

In step **454**, the user's client software, e.g. web browser plug-in, connects to the remote computer. The remote computer may have an application or agent operable to support the server-side operations corresponding to the client-side operations illustrated in FIG. **7**. Any of various application-level protocols may be used to communicate between the client software and the server software. In the preferred embodiment, a communication protocol based on the TCP/IP protocol is used for communication with the remote computer. At the time of connection, the remote graphical program may already be running on the remote computer, or the remote computer may be operable to launch the program in response to the client computer connecting.

In step **456**, the user's client software requests the remote computer to send a description of the user interface panel(s) associated with the graphical program specified in step **452**. Step **456** may be combined with step **454**. In response to this request, the remote computer sends the description of the user interface panel(s).

In step **458**, the user's client software, e.g. web browser plug-in, receives the description of the user interface panel(s) and displays the user interface panel(s) appropriately. In the preferred embodiment, the user interface panel description(s) that the client software receives is a description based on or identical to the description that the remote computer uses to persistently store the user interface panel information. In other words, when a graphical program and its user interface panel is created and saved on the remote computer, the information describing the user interface panel is structured in a particular way. In the preferred embodiment, the user's client software is operable to parse this structured information and display the user interface panel(s) appropriately on the user's display screen, e.g. in the window of the user's web browser.

It is noted, however, that in alternative embodiments the remote computer may transform the user interface panel description(s) before sending the description(s) to the client computer. For example, the user interface panel(s) may be stored on the remote computer in a binary form, but may be translated into a text form, e.g., a markup language description, which the client computer is operable to process in order to display the panel(s) appropriately. Such an embodiment may advantageously enable client computers with different types of display devices, e.g., small screens included in various types of wireless devices, to easily interpret and display the user interface panel description(s) differently, depending on the capabilities of the particular display devices.

In step **460**, the user's client software may receive data updates from the remote computer and update the user interface panel display accordingly. For example, as described above, the graphical program may be associated with measuring data from a live data source, and may be operable to display live data on the user interface panel continuously or periodically. Any of various data protocols may be used in transferring and displaying data updates.

The above description of step **460** pertains to an embodiment in which the user interface panel displayed on the client computer is "separated" from the actual data displayed in the panel. That is, the client computer may receive data to be displayed in the user interface panel independently of the panel description itself and may update the display of the panel according to the data, to reflect the output of the remote graphical program. In an alternative embodiment, the program output may be coupled with the panel description. For example, the panel description may be received as an image which reflects the program output. Thus, when receiving data updates, the client computer may receive an updated description of the user interface panel and may redisplay the updated panel.

In step **462**, the user may operate the user interface panel, e.g. by performing a GUI-style point and click operation. The user's client software brokers this GUI operation to the remote computer **86**. For example, as described above, the user's client software may communicate with a server-side agent, which may then forward the command to the remote graphical program. The remote graphical program then responds to the command accordingly. In many cases, the user's command in step **462** would cause the graphical program to change its output display, which would then be reflected on the user's display screen. In other words, in response to the user manipulating the inputs on the user interface displayed on the client computer **82**, the user input is provided to the graphical program executing on the server computer **86**, which may affect the displayed output of the graphical program. This displayed output is provided from the server computer **86** to be displayed on the user interface displayed on the client computer **82**. The user may then provide other input to the graphical user interface, and so on. Thus, steps **460** and **462** may be performed in an iterative manner.

Data Socket

In an alternative embodiment, the DataSocket system and method, disclosed in U.S. patent application Ser. No. 09/185,161, may be used to facilitate the communication between the user's client software and the remote computer running the graphical program. The DataSocket system comprises a client software component that addresses data sources/targets using a URL, much the way that a URL is used to address web pages anywhere in the world. When reading from an input

source, the DataSocket performs all work necessary to read the raw data from various input sources and to parse the data and return it in a form directly usable by the user's applications. For example, with respect to one embodiment of the present invention, the DataSocket may be used to receive a description of the remote graphical program's user interface panel and broker this description to, for example, a web browser plug-in operable to display the user interface panel in a web browser window. Once the user interface panel is displayed, the DataSocket may then receive data updates from the remote graphical program, which are displayed in the user interface panel.

When writing to an output target the Data Socket performs all work necessary to format the data provided by the user into the appropriate raw format for the specific target. For example, with respect to one embodiment of the present invention, the DataSocket may marshal the user's input commands into an appropriate format and send them to the remote graphical program. For more information on the DataSocket system and method, please refer to the above-referenced patent application.

Receiving the Block Diagram of the Remote Graphical Program

In one embodiment, a user may also request and receive the remote graphical program's block diagram. The block diagram may be displayed as a simple, non-interactive image that may be useful, for example, for the user to understand how the graphical program is implemented. For example, the remote graphical program may execute within a graphical programming environment that provides an ability to programmatically edit the graphical program. Thus, the user may use the information gained from the display of the block diagram to remotely edit the graphical program. For more information on dynamically creating or editing a graphical program, please refer to the above-referenced patent application Ser. No. 09/518,492 titled, "System and Method for Programmatically Creating a Graphical Program".

In another embodiment, the client computer may receive and view the actual block diagram, thereby enabling the user to view and edit the block diagram, using software on the client computer. The user of the client computer may then transfer the edited block diagram back to the server computer.

In another embodiment, the user may interactively perform operations such as program debugging while the graphical program executes on the remote computer. The client software may communicate with the remote computer in order to specify debugging information, such as break points, and to control program execution, such as continuing execution from break points, etc. The client software may be operable to illustrate the operation of the remote graphical program in various ways, e.g., by using execution highlighting to update the block diagram appearance to show real-time execution or data flow, etc.

FIGS. **8-10**: Exemplary Graphical Programs

FIGS. **8-10** illustrate several exemplary graphical programs to which the present system and method may be applied. Each figure illustrates a block diagram for the program and an associated user interface panel. As described above, the graphical program may execute on one computer, while one or more end users remotely view or interact with the user interface panel of the graphical program from a different computer. Also, an end user may remotely view and/or edit the block diagram of the graphical program. Each graphical program example is briefly described below.

The block diagram shown in FIG. **8**B simulates a tank control application. The associated user interface panel of FIG. **8**A displays a history of inflow, level, and temperature for the tank control application.

The block diagram shown in FIG. **9**B simulates an application that uses GPIB instruments to perform a frequency response test on a unit under test (UUT). A function generator supplies a sinusoidal input to the UUT (a bandpass filter in this example), and a digital multimeter measures the output voltage of the UUT.

The block diagram shown in FIG. **10**B simulates a temperature analysis application. This program reads a simulated temperature, sends an alarm if it is outside a given range, and determines a statistical mean, standard deviation, and histogram of the temperature history.

The example graphical programs shown in FIGS. **8-10** are directed toward instrumentation, industrial automation, or process control applications. The user interface panels for these programs include various controls or display readouts similar to what may appear on a hardware instrument or console. However, as discussed above, program developers and end users working in many different fields may benefit from the system and method described herein, to enable distributed display and/or control of a graphical program user interface for any of various types of applications.

Although the system and method of the present invention has been described in connection with the preferred embodiment, it is not intended to be limited to the specific form set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents, as can be reasonably included within the spirit and scope of the invention as defined by the appended claims.

We claim:

1. A non-transitory computer accessible memory medium that stores program instructions executable on a server computer to perform:

receiving user-provided information indicating a graphical program from a client computer, wherein the graphical program includes a block diagram that comprises a plurality of interconnected function icons representing graphical data flow of a desired function that visually indicate functionality of the graphical program, and wherein the user-provided information is received by the server computer over a network;

executing the graphical program, wherein said executing the graphical program comprises executing the plurality of interconnected function icons of the block diagram of the graphical program; and

providing information describing data updates of the block diagram of the graphical program to the client computer over the network during said executing, wherein the information describing the data updates of the block diagram of the graphical program is useable by the client computer to update a display of the block diagram of the graphical program on the client computer during said executing to reflect the graphical program executing on the server computer, and wherein the updated display of the block diagram of the graphical program is usable by a user of the client computer to analyze the graphical program executing on the server computer in real-time.

2. The non-transitory computer accessible memory medium of claim **1**, wherein the program instructions are further executable to perform:

receiving information from the client computer over the network specifying debugging of the graphical program; and

modifying the graphical program and/or execution of the graphical program on the server computer in response to the information specifying debugging.

3. The non-transitory computer accessible memory medium of claim 2, further comprising:

receiving at least one user-specified edit to the block diagram of the graphical program from the client computer over the network; and

editing the block diagram of the graphical program according to the at least one user-specified edit.

4. The non-transitory computer accessible memory medium of claim 2, wherein the information describing the data updates of the block diagram of the graphical program is useable by the client computer to update the display of the block diagram of the graphical program on the client computer to illustrate operations of the graphical program using execution highlighting to update the displayed block diagram to show real-time execution of the graphical program.

5. The non-transitory computer accessible memory medium of claim 1, wherein the program instructions are further executable to perform:

providing debugging results to the client computer over the network during said executing, wherein the debugging results are useable for display on the client computer during said executing.

6. The non-transitory computer accessible memory medium of claim 1, wherein the user-provided information indicating the graphical program comprises a uniform resource locator (URL) specifying one or more of:

the server computer; or

the graphical program on the server computer.

7. The non-transitory computer accessible memory medium of claim 1, wherein the program instructions are further executable to perform:

providing information to the client computer indicating a plurality of graphical programs on the server computer;

wherein the user-provided information indicating the graphical program comprises a user selection of the graphical program from the plurality of graphical programs.

8. The non-transitory computer accessible memory medium of claim 1, wherein the program instructions are further executable to perform:

receiving user manipulated inputs of at least one input variable of the graphical program over the network from the client computer;

executing the block diagram of the graphical program using the user manipulated inputs of at least one input variable to generate an output of at least one output variable; and

providing the output of at least one output variable over the network to the client computer for display on the client computer.

9. The non-transitory computer accessible memory medium of claim 1, wherein the graphical program comprises one or more of:

a graphical data flow program;

a graphical control flow program; or

a graphical execution flow program.

10. A method for executing a graphical program on a server computer and providing a block diagram of the graphical program on a client computer, the method comprising:

receiving a user-provided specification of the graphical program on the server computer from the client computer over a network, wherein the graphical program includes a block diagram that comprises a plurality of interconnected function icons representing graphical data flow of a desired function that visually indicate functionality of the graphical program;

executing the graphical program on the server computer, wherein said executing the graphical program comprises executing the plurality of interconnected function icons of the block diagram of the graphical program; and

providing information describing data updates of the block diagram of the graphical program to the client computer over the network during said executing, wherein the information describing the data updates of the block diagram of the graphical program is useable by the client computer to update a display of the block diagram of the graphical program on the client computer during said executing to reflect the graphical program executing on the server computer, and wherein the updated display of the block diagram of the graphical program is usable by a user of the client computer to analyze the graphical program executing on the server computer in real-time.

11. A method for displaying a remotely executing block diagram of a graphical program on a client computer, the method comprising:

receiving user input to the client computer, wherein the user input indicates the graphical program on a server computer, and wherein the graphical program includes a block diagram that comprises a plurality of interconnected function icons representing graphical data flow of a desired function that visually indicate functionality of the graphical program;

providing the indication of the graphical program to the server computer over a network;

executing the graphical program on the server computer, wherein said executing the graphical program comprises executing the plurality of interconnected function icons of the block diagram of the graphical program;

receiving information describing data updates of the block diagram of the graphical program from the server computer over the network during said executing; and

updating a display of the block diagram of the graphical program on the client computer during said executing to reflect the graphical program executing on the server computer using the information describing the data updates of the block diagram of the graphical program by the client computer;

wherein the updated display of the block diagram of the graphical program is usable by a user of the client computer to analyze the graphical program executing on the server computer in real-time.

12. The method of claim 11, further comprising:

sending debugging information over the network to the server computer specifying modification of the graphical program and/or execution of the graphical program on the server computer.

13. The method of claim 12, further comprising:

receiving debugging results from the server computer over the network during said executing; and

displaying the debugging results on a display of the client computer during said executing.

14. The method of claim 12, further comprising:

receiving user input specifying an edit to the block diagram of the graphical program; and

providing the user input specifying the edit to the server computer over the network;

wherein the server computer is operable to edit the block diagram of the graphical program according to the user input specifying the edit.

15. The method of claim 12, wherein said updating the display of the block diagram of the graphical program comprises illustrating operations of the graphical program using

execution highlighting to update the displayed block diagram to show real-time execution of the graphical program.

16. The method of claim **12**, further comprising:

connecting to the server computer over the network after said receiving the user input;

wherein said receiving information describing the data updates of the block diagram of the graphical program is performed after said receiving the user input and said connecting; and

wherein the graphical program is already executing on the server computer when said connecting occurs, or execution of the graphical program is launched in response to said connecting.

17. The method of claim **12**, wherein said updating the display of the block diagram of the graphical program comprises updating the display of the block diagram of the graphical program on a web browser.

18. The method of claim **12**, wherein the user input indicating the graphical program comprises a uniform resource locator (URL) specifying one or more of:

the server computer; or

the graphical program on the server computer.

19. The method of claim **12**, further comprising:

displaying information indicating a plurality of graphical programs on the server computer;

wherein the user input indicating the graphical program selects the graphical program from the plurality of graphical programs.

20. The method of claim **12**, further comprising:

receiving user manipulated inputs of at least one input variable for the graphical program;

providing the user manipulated inputs of at least one input variable to the server computer over the network, wherein the execution of the graphical program comprises the block diagram of the graphical program executing using the user manipulated inputs of at least one input variable and generating an output of at least one output variable;

receiving the output of at least one output variable over the network; and

displaying the output of at least one output variable on a display of the client computer.

* * * * *